Cours d'informatique

MPSI-PCSI

Outman El Hichami

CPGE - Tanger

2017/2018



Plan

Notion de l'informatique



- Informatique : il est composé des deux mots : information et automatique.
- Informatique : ensemble des disciplines scientifiques du traitement automatique de l'information, à l'aide des machines automatiques.



Notion de l'ordinateur

Comment et où les ordinateurs sont utilisés ?

Les ordinateurs jouent un rôle important et indispensable dans la vie de chaque jour. Ils couvrent tous les types d'environnements :

- Entreprise,
- Chez-soi,
- Hôpitaux et centres médicaux,
- Ecoles, universités,
- Voitures, avions,
- ...



Notion de l'ordinateur





- L'ordinateur est une machine électronique de traitement de l'information capable d'exécuter un ensemble d'instructions et de stocker n'importe quelles données en un temps très court et sans risque d'erreurs.
- Une tablette numérique, un téléphone portable (smartphone),... sont des ordinateurs, au même titre qu'un ordinateur de bureau.

Notion de l'ordinateur



Pour effectuer le traitement automatiquement des informations, 2 choses doivent cohabiter (travaillent ensemble) :

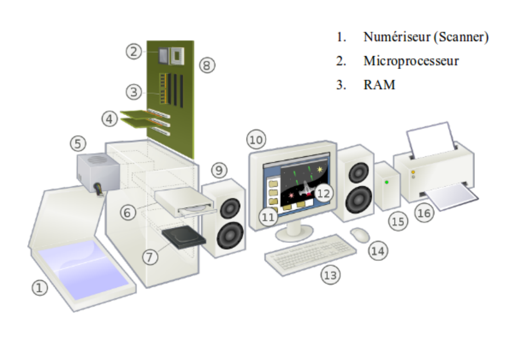
Matériels

Composantes physiques internes ou externes d'un ordinateur.

Logiciels

- Système d'exploitation : Ensemble de programmes qui pilotent les matériels d'un ordinateur (Windows,...).
- Logiciels d'application : Programmes chargées dans l'ordinateur pour réaliser une fonction précise (Word,...).

Environnement matériel et logiciel



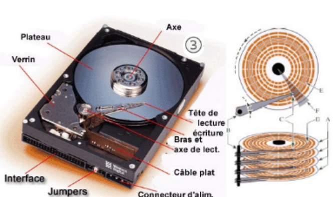
1. Numériseur (Scanner)
2. Microprocesseur
3. RAM
4. Cartes d'extension
5. Boite d'alimentation
6. Lecteur de disque
7. Disque dur
8. Carte mère
9. Haut parleur
10. Moniteur (écran)
11. Système d'exploitation
12. Application (Logiciel)
13. Clavier
14. Souris
15. Onduleur
16. Imprimante

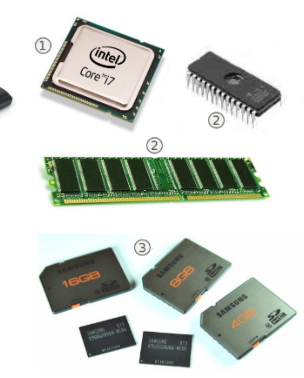
Périphériques



Périphériques

1. Microprocesseur
2. Mémoire vive et morte
3. Mémoire de masse
4. Périphériques et ports





A noter :



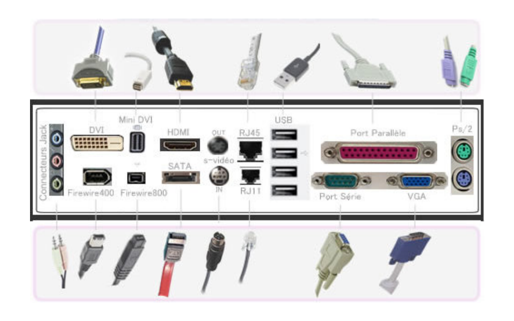
- **Microprocesseur** : Circuit électronique complexe qui exécute des instructions en quelques cycles d'horloges (unité élémentaire de temps d'un ordinateur).
- **Mémoire vive** : C'est la mémoire **RAM** ; Random Access Memory (Mémoire à accès aléatoire). C'est une mémoire en lecture-écriture. Mémoire volatile car toutes les données sont perdues à l'élimination de l'alimentation électrique.
- **Mémoire morte** : C'est la mémoire **ROM** ; Read-Only Memory. C'est une mémoire informatique non volatile, contient des programmes (**BIOS**) nécessaires au démarrage d'un ordinateur.

A noter :



- **Mémoire de masse** : Mémoire de grande capacité, non volatile et qui peut être lue et écrite par un ordinateur. Par exemple : Disque dur, CD-ROM, DVD-ROM, Disque externe.
- **Périphériques** : Dispositifs connectés à un ordinateur qui servent à y fournir des informations (entrée) ou/et y faire sortir des informations (sortie). On peut également rencontrer des périphériques d'entrée-sortie qui opèrent dans les deux sens : un graveur de CD-ROM ...

Ports et Connecteurs





Les connecteurs informatiques (On dit également : Port, interface ou encore, connecteurs d'entrée-sortie), sont des liaisons permettant de relier à l'ordinateur des équipements ou périphériques en utilisant des câbles ou des liaisons sans fil.

A noter :



- Port **USB** : L'Universal Serial Bus. Le bus USB permet de connecter des périphériques en bénéficiant du **Plug and Play** (permettant aux périphériques d'être reconnus rapidement et automatiquement par le système d'exploitation dès le branchement du matériel, et sans redémarrage de l'ordinateur).
- Port **PS/2** : Sigle de Personal System/2, est un port de connexion de dimensions réduites pour souris ou clavier.

A noter :



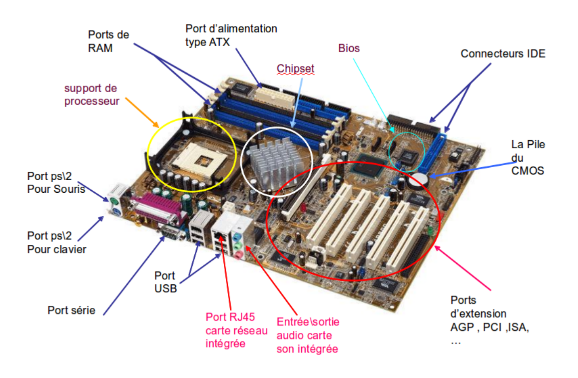
- Port **Rj45** : Une des principaux connecteurs de carte réseau pour les réseaux.
- Port **Rj11** : Le connecteur le plus utilisé pour les lignes téléphoniques.

A noter :



- Port **VGA** : Ce type de connecteur équipe notamment la plupart des cartes graphiques en permettant d'envoyer à l'écran 3 signaux analogiques correspondant aux composantes rouges, bleues et vertes de l'image.
- Port **HDMI** : (High Definition Multimedia Interface) est une interface numérique permettant le transfert de données multimédia (audio et vidéo).
- ...

Carte mère





Carte mère : Carte électronique qui permet aux différents composants de communiquer via différents **bus de communication**.

A noter :



- **Chipset** : Pour gérer les flux de données numériques entre le processeur, la mémoire et les périphériques.
- Connecteurs **SATA** : Pour brancher les disques durs, lecteurs et graveurs.
- Ports d'extension **AGP, PCI, ISA, ...** : Pour accueillir des cartes d'extension et d'ajouter des capacités ou des fonctionnalités à un ordinateur.
- Pile **CMOS** : Permettre l'ordinateur de rester a l'heure et à la date exacte même débranché.

Définition





Système d'exploitation S.E. : est un ensemble de logiciels (programmes) qui permettent l'exploitation d'une machine informatique. Exemples : Windows, Unix, ...

Objectifs





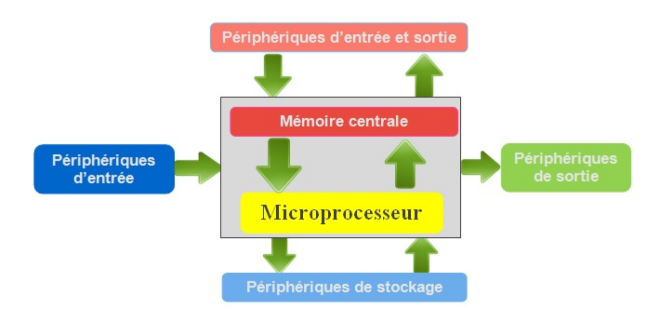
- Gestion de l'enchaînement des tâches en tenant compte des priorités.
- Gestion des Entrées/Sorties (lecture et écriture, impression, affichage sur écran...)
- Gestion des mémoires et de microprocesseur.
- Prise en charge d'éventuelles erreurs détectées.

Les types d'un système d'exploitation



- S.E. **Mono-tâche** : On ne peut exécuter qu'un seul programme à la fois.
- S.E. **Multi-tâches** : Lorsque plusieurs programmes peuvent être exécutées simultanément.
- S.E. **Mono-utilisateur** : Lorsqu'un seul utilisateur peut utiliser l'ordinateur.
- S.E. **Multi-utilisateurs** : Lorsque plusieurs utilisateurs peuvent manipuler simultanément l'ordinateur.

Schéma fonctionnel





Ordinateur = Machine qui saisit (**périphériques d'entrée**), stocke (**mémoire**), traite (**programmes**) et restitue (**périphériques de sortie**) des informations.

Notions de codage

Exemple 1

Nous utilisons le codage chaque jour :

- 11,
- onze,
- XI,
- ...

Mais c'est toujours le même nombre.

Notions de codage



Exemple 2

Si on demande le nom correspondant à l'image :

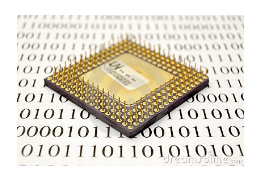
- À un arabophone, il répond : بيت
- À un francophone, il répond : maison
- À un anglo-saxon, il répond : house
- ...

Mais c'est toujours la même image, seule le nom la désignant change.

Notions de codage

Comment l'information est représentée dans l'ordinateur ?

Codage binaire





- Les mémoires mémorisent, transmettent et transforment des données (nombres, textes, images, sons, ...) par une suite de chiffres binaires (**bits**).
- Un bit ne peut prendre que deux valeurs : 0 et 1.
- En pratique la mémoire est organisée en **octets** (paquets de 8 bits).

Codage binaire : Entiers naturels



- Le système binaire est un système de numération utilisant la base 2.
- Par exemple le nombre décimal $(243)_{10}$ se note : $(11\ 110\ 011)_2$
- Sa décomposition est la suivante...

Codage binaire : Entiers naturels

243 | 2
1 | 121 | 2
1 | 60 | 2
0 | 30 | 2
0 | 15 | 2
1 | 7 | 2
1 | 3 | 2
1 | 1 | 2
1 | 0

Sens de lecture



- En faisant des divisions entières successives par 2, on obtient une suite de restes (en bleu).
- En lisant les restes du bas vers le haut on obtient : $(11\ 110\ 011)_2$.

Codage binaire : Entiers naturels



- La numération décimale peut être construite à partir de la numération binaire en utilisant cette méthode :
- Par exemple, à partir de cette représentation $(11\ 110\ 011)_2$ on obtient le nombre décimal :
- $(1 \text{ x } 2^7) + (1 \text{ x } 2^6) + (1 \text{ x } 2^5) + (1 \text{ x } 2^4) + (0 \text{ x } 2^3) + (0 \text{ x } 2^2) + (1 \text{ x } 2^1) + (1 \text{ x } 2^0) = 128 + 64 + 32 + 16 + 0 + 0 + 2 + 1 = (243)_{10}$

Codage binaire : Entiers naturels

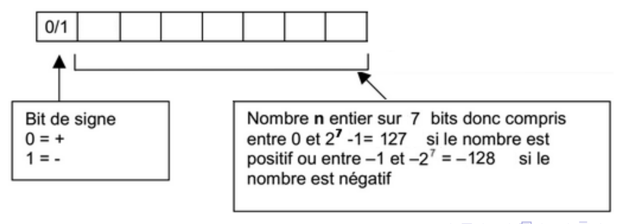
Exercice

- Trouver la représentation en base binaire de : $(67)_{10}$, $(128)_{10}$ et $(566)_{10}$.
- Trouver la représentation en base décimale de : $(11\ 101\ 010)_2$ et $(1\ 010\ 101\ 010\ 100\ 001)_2$.

Codage binaire : Entiers signés



- Un entier signé sera représenté en binaire comme un entier naturel.
- La seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe.
- (0 correspond à un signe positif et 1 à un signe négatif).



Codage binaire : Entiers signés



Méthode de codage

Un entier signé sera représenté grâce au codage en **complément à deux**.

Codage binaire : Entiers signés



Exemple

On désire coder la valeur $(-19)_{10}$ sur 8 bits. Il suffit :

- d'écrire $(|-19|)_{10}$ en binaire : $(19)_{10} = (00\ 010\ 011)_2$
- d'écrire son **complément à 1** : $(\overline{19})_{10} = (11\ 101\ 100)_2$
- et d'ajouter 1 : $+ \quad 1$

complément à 2 $= (11\ 101\ 101)_2$

La représentation binaire de $(-19)_{10}$ sur 8 bits est : $(11\ 101\ 101)_2$.

Codage binaire : Entiers signés



Exemple

On désire coder la valeur $(\text{-}19)_{10}$ sur 8 bits. Il suffit :

- d'écrire $(|\text{-}19|)_{10}$ en binaire : $(19)_{10} = (00\ 010\ 011)_2$
- d'écrire son **complément à 1** : $(\overline{19})_{10} = (11\ 101\ 100)_2$
- et d'ajouter 1 : $+ \quad 1$

complément à 2 $= (11\ 101\ 101)_2$

La représentation binaire de $(\text{-}19)_{10}$ sur 8 bits est : $(11\ 101\ 101)_2$.

Codage binaire : Entiers signés



Exemple

On désire coder la valeur $(\text{-}19)_{10}$ sur 8 bits. Il suffit :

- d'écrire $(|\text{-}19|)_{10}$ en binaire : $(19)_{10} = (00\ 010\ 011)_2$
- d'écrire son **complément à 1** : $(\overline{19})_{10} = (11\ 101\ 100)_2$
- et d'ajouter 1 : $+ \quad 1$

complément à 2 $= (11\ 101\ 101)_2$

La représentation binaire de $(\text{-}19)_{10}$ sur 8 bits est : $(11\ 101\ 101)_2$.

Codage binaire : Entiers signés

Conversion d'un nombre binaire négatif en décimal

- **Exemple** : soit à convertir $(11\ 101\ 101)_2$ en décimal
- **Méthode** : **complément à 2**
  - $(11\ 101\ 101)_2$
  - **complément à 1** : $(00\ 010\ 010)_2$
  - **complément à 2** : $(00\ 010\ 011)_2$
  - $(00\ 010\ 011)_2 = (19)_{10}$
  - $(11\ 101\ 101)_2 = (-19)_{10}$

Codage binaire : Entiers signés



Conversion d'un nombre binaire négatif en décimal

- **Exemple** : soit à convertir $(11\ 101\ 101)_2$ en décimal
- **Méthode** : **complément à 2**
  - $(11\ 101\ 101)_2$
  - **complément à 1** : $(00\ 010\ 010)_2$
  - **complément à 2** : $(00\ 010\ 011)_2$
  - $(00\ 010\ 011)_2 = (19)_{10}$
  - $(11\ 101\ 101)_2 = (-19)_{10}$

Codage binaire : Entiers signés

Exercice

- Codez les entiers signés suivants sur 8 bits (16 si nécessaire) : $(-56)_{10}$, $(-233)_{10}$, $(456)_{10}$.
- Que valent en base dix les trois entiers signés suivants : $(01\ 101\ 100)_2$, $(11\ 001\ 101)_2$ et $(1\ 010\ 101\ 010\ 101\ 010)_2$?

Codage binaire : Nombres réels : Virgule fixe



- En base 10, l'expression $(652{,}375)_{10}$ signifie :
  $6x10^2 + 5x10^1 + 2x10^0 + 3x10^{-1} + 7x10^{-2} + 5x10^{-3}$
- De même pour la base 2, l'expression $(110{,}101)_2$ signifie :
  $1x2^2 + 1x2^1 + 0 \text{ x } 2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

Exemple : On peut ainsi facilement convertir un nombre réel de la base 2 vers la base 10 :

$$(110{,}101)_2 = 1x2^2 + 1x2^1 + 0 \text{ x } 2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$$
$$= 4 + 2 + 0{,}5 + 0{,}125 = (6{,}625)_{10}.$$



Codage binaire : Nombres réels : Virgule fixe

Exercice

- Transformez $(0{,}010\ 101\ 010\ 1)_2$ et $(11\ 100{,}100\ 01)_2$ en base 10.

Codage binaire : Nombres réels : Virgule fixe



Conversion de décimal en binaire

Par exemple : convertissons $(1234{,}347)_{10}$ en base 2.

- La partie entière se transforme comme : $(1234)_{10} = (10\ 011\ 010\ 010)_2$
- On transforme la partie décimale selon le schéma suivant :
  - $0{,}347 \times 2 = \mathbf{0}{,}694 \Rightarrow (0{,}347)_{10} = (0{,}\mathbf{0}\ldots)_2$
  - $0{,}694 \times 2 = \mathbf{1}{,}388 \Rightarrow (0{,}347)_{10} = (0{,}0\mathbf{1}\ldots)_2$
  - $0{,}388 \times 2 = \mathbf{0}{,}766 \Rightarrow (0{,}347)_{10} = (0{,}01\mathbf{0}\ldots)_2$
  - $0{,}766 \times 2 = \mathbf{1}{,}552 \Rightarrow (0{,}347)_{10} = (0{,}010\mathbf{1}\ldots)_2$
  - $0{,}552 \times 2 = \mathbf{1}{,}104 \Rightarrow (0.347)_{10} = (0{,}0101\mathbf{1}\ldots)_2$
- $\Rightarrow (1234{,}347)_{10} = (10\ 011\ 010\ 010{,}010\ 11\ldots)_2$

Codage binaire : Nombres réels : Virgule fixe



Conversion de décimal en binaire

Par exemple : convertissons $(1234{,}347)_{10}$ en base 2.

- La partie entière se transforme comme : $(1234)_{10} = (10\ 011\ 010\ 010)_2$
- On transforme la partie décimale selon le schéma suivant :
  - $0{,}347 \times 2 = \mathbf{0}{,}694 \Rightarrow (0{,}347)_{10} = (0{,}\mathbf{0}...)_2$
  - $0{,}694 \times 2 = \mathbf{1}{,}388 \Rightarrow (0{,}347)_{10} = (0{,}0\mathbf{1}...)_2$
  - $0{,}388 \times 2 = \mathbf{0}{,}766 \Rightarrow (0{,}347)_{10} = (0{,}01\mathbf{0}...)_2$
  - $0{,}766 \times 2 = \mathbf{1}{,}552 \Rightarrow (0{,}347)_{10} = (0{,}010\mathbf{1}...)_2$
  - $0{,}552 \times 2 = \mathbf{1}{,}104 \Rightarrow (0.347)_{10} = (0{,}0101\mathbf{1}...)_2$
- $\Rightarrow (1234{,}347)_{10} = (10\ 011\ 010\ 010{,}010\ 11...)_2$

Codage binaire : Nombres réels : Virgule fixe



Conversion de décimal en binaire

Par exemple : convertissons $(1234{,}347)_{10}$ en base 2.

- La partie entière se transforme comme : $(1234)_{10} = (10\ 011\ 010\ 010)_2$
- On transforme la partie décimale selon le schéma suivant :
  - $0{,}347 \times 2 = \mathbf{0}{,}694 \Rightarrow (0{,}347)_{10} = (0{,}\mathbf{0}...)_2$
  - $0{,}694 \times 2 = \mathbf{1}{,}388 \Rightarrow (0{,}347)_{10} = (0{,}0\mathbf{1}...)_2$
  - $0{,}388 \times 2 = \mathbf{0}{,}766 \Rightarrow (0{,}347)_{10} = (0{,}01\mathbf{0}...)_2$
  - $0{,}766 \times 2 = \mathbf{1}{,}552 \Rightarrow (0{,}347)_{10} = (0{,}010\mathbf{1}...)_2$
  - $0{,}552 \times 2 = \mathbf{1}{,}104 \Rightarrow (0.347)_{10} = (0{,}0101\mathbf{1}...)_2$
- $\Rightarrow (1234{,}347)_{10} = (10\ 011\ 010\ 010{,}010\ 11...)_2$

Codage binaire : Nombres réels : Virgule fixe

Exercice

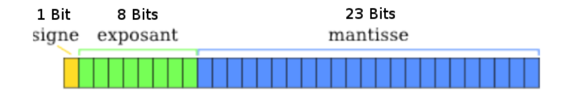
- Transformez $(12{,}910)_{10}$, $(0{,}562510)_{10}$ et $(0{,}1510)_{10}$ en base 2.

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Cette norme propose de coder un nombre sur 32 bits et définit trois composantes :

- Le **Signe** est représenté par un seul bit, le bit de poids fort,
- L'**Exposant** est codé sur les 8 bits consécutifs au signe,
- La **Mantisse** (bits situés après la virgule) sur les 23 bits restants.



Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- **Exposant** = 127 + 5 = $(132)_{10}$ = $(10\ 000\ 100)_2$
- $(40)_{10}$ = $(0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- **Signe** : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  $101\ 000 = \mathbf{1{,}010\ 00}\text{x}2^5$ (2^5 décale la virgule de 5 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- **Exposant** $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : 1,mantisse
  $101\ 000 = 1{,}010\ 00 \text{x} 2^5$ (2^5 décale la virgule de 5 chiffres vers la droite)
- Mantisse étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- Exposant $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- **Signe** : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- **Exposant** = 127 + 5 = $(132)_{10}$ = $(10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- Mantisse étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- Exposant $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- Exposant $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- **Exposant** $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)

Exemple : Traduisons en binaire, le nombre $(40)_{10}$

- Signe : $+ \Rightarrow 0$
- Codons la valeur absolue en binaire : $(40)_{10} = (101\ 000)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  101 000 = **1,010 00**x2^5 (2^5 décale la virgule de 5 chiffres vers la droite)
- Mantisse étendue sur 23 bits est donc :
  010 00 000 000 000 000 000 000
- Exposant $= 127 + 5 = (132)_{10} = (10\ 000\ 100)_2$
- $(40)_{10} = (0\ 10\ 000\ 100\ 010\ 00\ 000\ 000\ 000\ 000\ 000\ 000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Exercice

- Exprimez les nombres $(13)_{10}$ et $(-6{,}625)_{10}$ en utilisant la norme IEEE 754

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Exercice Pour $(\mathbf{13})_{10}$:

- **Signe** : $+ \Rightarrow 0$
- Codons en binaire : $(13)_{10} = (1101{,}1010)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  $1101 = \mathbf{1{,}101}\text{x}2^3$ (2^3 décale la virgule de 3 chiffres vers la droite)
- **Mantisse** étendue sur 23 bits est donc :
  101 0000 0000 0000 0000 0000
- **Exposant** $= 127 + 3 = (130)_{10} = (10000010)_2$
- $(13{,}625)_{10} = (\mathbf{0}\ \mathbf{10000010}\ \mathbf{101}\ \mathbf{0000}\ \mathbf{0000}\ \mathbf{0000}\ \mathbf{0000}\ \mathbf{0000})_2$

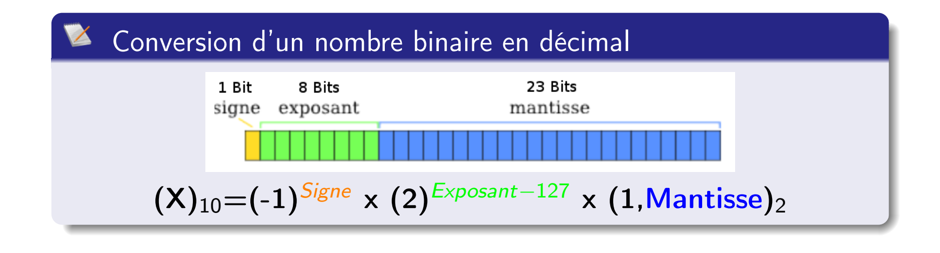
Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Exercice Pour $(\textbf{-6,625})_{10}$:

- Signe : - $\Rightarrow$ 1
- Codons la valeur absolue en binaire : $(6{,}625)_{10} = (110{,}1010)_2$
- Nous mettons ce nombre sous la forme : **1,mantisse**
  $110{,}1010 = 1{,}\mathbf{101010}\text{x}2^2$
- Mantisse étendue sur 23 bits est donc :
  101010 0000 0000 0000 00000
- Exposant $= 127 + 2 = (129)_{10} = (10000001)_2$
- $(-6{,}625)_{10} = (1\ 10000001\ 101010\ 0000\ 0000\ 0000\ 00000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



$$(X)_{10} = (-1)^{Signe} \times (2)^{Exposant-127} \times (1,Mantisse)_2$$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Exercice

Traduisez en décimal le nombre :

$(0\ 10000010\ 0100100\ 0000\ 0000\ 0000\ 0000)_2$

Codage binaire : Nombres réels : La norme IEEE 754 (Virgule flottante)



Exercice

Pour (0 10000010 0100100 0000 0000 0000 0000)$_2$:

- **Signe** : 0 ⇒ positif
- **Exposant** = $(10000010)_2$ ⇒ 130 - 127 = 3
- **Mantisse** = 0100100 0000 0000 0000 0000 ⇒ $(1{,}01001)_2$
- **Résultat** = $+(1{,}01001)_2 \times 2^3 = +(1010{,}01)_2 = +(10{,}25)_{10}$

Conversion du Décimal vers une base B



- Pour représenter un nombre A décimal dans une autre base B, on divise A par B, puis on divise le quotient obtenu par B, et ainsi de suite jusqu'a l'obtention d'un quotient nul.
- La suite des restes obtenus correspond aux chiffres dans la base B visée :

A | B
a_0 | Q_1 | B
a_1 | Q_2 | B
a_2 | Q_3 | B
a_3 | 0

Restes successifs

$$(A)_{10} = (a_3a_2a_1a_0)_B$$

Conversion du Décimal vers une base B

Représentation en base dix un entier naturel A donné en base B

Si $(a_m a_{m-1} ... a_0)_B$ est l'écriture de A en base B, alors :

$$(A)_{10} = \sum_{i=0}^{m} a_i x B^i = a_0 x B^0 + a_1 x B^1 + a_2 x B^2 + \ldots + a_m x B^m.$$

Représentation en base dix un nombre réel positif A donné en base B

Si $(a_m a_{m-1} ... a_0, b_1 b_2 ... b_n)_B$ est l'écriture de A en base B, alors :

$$(A)_{10} = \sum_{i=0}^{m} a_i x B^i + \sum_{j=1}^{n} b_j x B^{-j} = a_0 x B^0 + a_1 x B^1 + a_2 x B^2 + \ldots + a_m x B^m + b_1 x B^{-1} + b_2 x B^{-2} + \ldots + b_n x B^{-n}.$$

Conversion du Décimal vers une base B

Représentation en base dix un entier naturel A donné en base B

Si $(a_m a_{m-1} ... a_0)_B$ est l'écriture de A en base B, alors :

$$(A)_{10} = \sum_{i=0}^{m} a_i x B^i = a_0 x B^0 + a_1 x B^1 + a_2 x B^2 + \ldots + a_m x B^m.$$

Représentation en base dix un nombre réel positif A donné en base B

Si $(a_m a_{m-1} ... a_0, b_1 b_2 ... b_n)_B$ est l'écriture de A en base B, alors :

$$(A)_{10} = \sum_{i=0}^{m} a_i x B^i + \sum_{j=1}^{n} b_j x B^{-j} = a_0 x B^0 + a_1 x B^1 + a_2 x B^2 + \ldots + a_m x B^m + b_1 x B^{-1} + b_2 x B^{-2} + \ldots + b_n x B^{-n}.$$

Codage Octal



- Le système de numération octal est le système de numération de base 8, il utilise donc les chiffres de 0 à 7 :
- $(2735)_8 = 2x8^3 + 7x8^2 + 3x8^1 + 5x8^0 = (1501)_{10}$
- $(171{,}3046875)_{10}$ se convertit en octal en calculant : ...
  - Conversion de $(171)_{10}$: donne $(253)_8$
  - Conversion de 0,3046875 :
    0,3046875 x 8 = 2,4375 = 2 + 0,4375
    0,4375 x 8 = 3,5 = 3 + 0,5
    0,5 x 8 = 4
  - $(171{,}3046875)_{10} = (253{,}234)_8$

Codage Octal



- Le système de numération octal est le système de numération de base 8, il utilise donc les chiffres de 0 à 7 :
- $(2735)_8 = 2x8^3 + 7x8^2 + 3x8^1 + 5x8^0 = (1501)_{10}$
- $(171{,}3046875)_{10}$ se convertit en octal en calculant : ...
  - Conversion de $(171)_{10}$: donne $(253)_8$
  - Conversion de 0,3046875 :
    $0{,}3046875 \times 8 = 2{,}4375 = 2 + 0{,}4375$
    $0{,}4375 \times 8 = 3{,}5 = 3 + 0{,}5$
    $0{,}5 \times 8 = 4$
  - $(171{,}3046875)_{10} = (253{,}234)_8$

Codage Octal



- Le système de numération octal est le système de numération de base 8, il utilise donc les chiffres de 0 à 7 :
- $(2735)_8 = 2x8^3 + 7x8^2 + 3x8^1 + 5x8^0 = (1501)_{10}$
- $(171{,}3046875)_{10}$ se convertit en octal en calculant : ...
  - Conversion de $(171)_{10}$: donne $(253)_8$
  - Conversion de 0,3046875 :
    $0{,}3046875 \times 8 = 2{,}4375 = 2 + 0{,}4375$
    $0{,}4375 \times 8 = 3{,}5 = 3 + 0{,}5$
    $0{,}5 \times 8 = 4$
  - $(171{,}3046875)_{10} = (253{,}234)_8$

Codage Hexadécimal



- Le système hexadécimal est un système de numération positionnel en base 16.
- Il utilise ainsi 16 symboles, en général les chiffres arabes pour les dix premiers chiffres et les lettres de A à F pour les six suivants.

Codage Hexadécimal



Exemple

- $(15AACF7)_{16}$ se convertit en décimal en calculant : ...
- $(15AACF7)_{16} = 1x16^6 + 5x16^5 + 10x16^4 + 10x16^3 + 12x16^2 + 15x16^1 + 7x16^0 = (22719735)_{10}$
- $(171{,}3046875)_{10}$ se convertit en hexadécimal en calculant : ...
  - Conversion de $(171)_{10}$: donne $(AB)_{16}$
  - Conversion de 0,3046875 :
    $0{,}3046875 \times 16 = 4{,}875 = 4 + 0{,}875$
    $0{,}875 \times 16 = 14{,}0 = 14 + 0$
  - $(171{,}3046875)_{10} = (AB{,}4E)_{16}$

Codage Octal et Hexadécimal



Exercice

- Convertir de $(110{,}6046875)_{10}$ en octal et hexadécimal.

Changement de base : de la base binaire à la base octal ou hexadécimal

Exemple : Binaire vers l'octal

- $(1010101010)_2$ à convertir en octal
- $\underbrace{001}\underbrace{010}\underbrace{101}\underbrace{010}$ (on regroupe par 3 à partir de la droite)
- 1 2 5 2 $\Rightarrow (1252)_8$

Exemple : Binaire vers l'hexadécimal

- $(1010101010)_2$ à convertir en hexadécimal
- $\underbrace{0010}\underbrace{1010}\underbrace{1010}$ (on regroupe par 4 à partir de la droite)
- 2 A A $\Rightarrow (2AA)_{16}$

Changement de base : de la base binaire à la base octal ou hexadécimal

Exemple : Binaire vers l'octal

- $(1010101010)_2$ à convertir en octal
- $\underbrace{001}\,\underbrace{010}\,\underbrace{101}\,\underbrace{010}$ (on regroupe par 3 à partir de la droite)
- 1 2 5 2 $\Rightarrow (1252)_8$

Exemple : Binaire vers l'hexadécimal

- $(1010101010)_2$ à convertir en hexadécimal
- $\underbrace{0010}\,\underbrace{1010}\,\underbrace{1010}$ (on regroupe par 4 à partir de la droite)
- 2 A A $\Rightarrow (2AA)_{16}$

Changement de base : de la base binaire à la base octal ou hexadécimal



Exercice

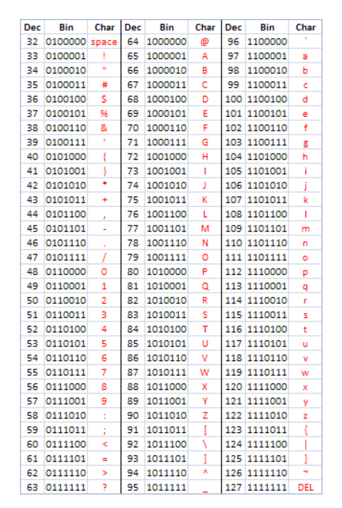
- Exprimez les nombres $(40)_{10}$ et $(\text{-}6{,}625)_{10}$ en octal et hexadécimal
- NB : Utilisez la norme IEEE 754 et puis le regroupement de bits.

La norme ASCII



- La norme ASCII est une norme de codage de caractères en informatique.
- La norme ASCII vise à donner à tout caractère de n'importe quel système d'écriture de langue (Arabe, Chinois, ...) un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plate-forme informatique ou le logiciel.

La norme ASCII



Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char
32	0100000	space	64	1000000	@	96	1100000	`
33	0100001	!	65	1000001	A	97	1100001	a
34	0100010	"	66	1000010	B	98	1100010	b
35	0100011	#	67	1000011	C	99	1100011	c
36	0100100	$	68	1000100	D	100	1100100	d
37	0100101	%	69	1000101	E	101	1100101	e
38	0100110	&	70	1000110	F	102	1100110	f
39	0100111	'	71	1000111	G	103	1100111	g
40	0101000	(	72	1001000	H	104	1101000	h
41	0101001	)	73	1001001	I	105	1101001	i
42	0101010	*	74	1001010	J	106	1101010	j
43	0101011	+	75	1001011	K	107	1101011	k
44	0101100	,	76	1001100	L	108	1101100	l
45	0101101	-	77	1001101	M	109	1101101	m
46	0101110	.	78	1001110	N	110	1101110	n
47	0101111	/	79	1001111	O	111	1101111	o
48	0110000	0	80	1010000	P	112	1110000	p
49	0110001	1	81	1010001	Q	113	1110001	q
50	0110010	2	82	1010010	R	114	1110010	r
51	0110011	3	83	1010011	S	115	1110011	s
52	0110100	4	84	1010100	T	116	1110100	t
53	0110101	5	85	1010101	U	117	1110101	u
54	0110110	6	86	1010110	V	118	1110110	v
55	0110111	7	87	1010111	W	119	1110111	w
56	0111000	8	88	1011000	X	120	1111000	x
57	0111001	9	89	1011001	Y	121	1111001	y
58	0111010	:	90	1011010	Z	122	1111010	z
59	0111011	;	91	1011011	[	123	1111011	{
60	0111100	<	92	1011100	\	124	1111100	|
61	0111101	=	93	1011101	]	125	1111101	}
62	0111110	>	94	1011110	^	126	1111110	~
63	0111111	?	95	1011111	_	127	1111111	DEL

La norme ASCII



Exercice

En utilisant le code ASCII :

- Ecrire votre nom et votre prénom sans oublier de mettre les initiales en majuscule.
- Décoder la phrase suivante : 76 39 97 114 99 104 105 44 32 99 39 101 115 116 32 102 97 99 105 108 101 46.
- Décoder la phrase suivante donnée en binaire : 1001010 0100111 1000001 1001001 0100000 1010100 1010010 1001111 1010101 1010110 1000101 0100000 0100001.

La norme RGB (Red, Green, Blue)



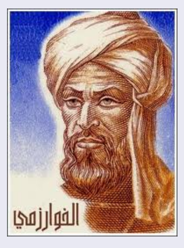
- Se présente comme un nombre hexadécimal à six chiffres : FF06C3 par exemple.
- Chaque paire de chiffres est dédiée à une couleur primaire. Sur le même exemple, cela donne :
  - FF pour le rouge ;
  - 06 pour le vert ;
  - C3 pour le bleu ;
- 000000 : noir.
- FFFFFF : blanc.



Introduction



Mais c'est quoi un algorithme ?



Muhammad ibn Musa al-Khuwarizmi (ca. 780 - 850)

Introduction



Exemple 1

Problème : L'accès aux CPGE

⇒ Les démarches à suivre :

1. titulaires d'un baccalauréat marocain ou étranger
2. avec, au moins, mention "assez bien"
3. à condition de ne pas dépasser "21 ans"
4. inscritpion sur le site eCPGE
5. photocopie légalisée du Baccalauréat
6. copies des relevés de notes de la 1ère et la 2ème année du baccalauréat
7. copie de la CNI (Carte Nationale d'Identité)



Introduction



Exemple 2

Problème : Résolution d'une équation du 2nd degré : $ax^2+bx+c=0$.

⇒ Etapes de résolution :

1. Connaître les valeurs de a, b et c
2. Calculer le discriminant $\Delta = b^2 - 4ac$
3. Si $\Delta < 0$ alors pas de solution réelle
4. Si $\Delta = 0$ alors une seule solution : $x = \frac{-b}{2a}$
5. Si $\Delta > 0$ alors deux solutions : $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$

Structure d'un algorithme



Exemple 3

Problème : Usage d'un appareil téléphonique à pièces de monnaie pour effectuer une communication.

Algorithme téléphone

Début

Décrocher l'appareil

Insérer les pièces nécessaires

Composer le numéro

Communiquer

Raccrocher

Fin



Structure d'un algorithme



Exemple 4

Problème : Recette de cuisine (Tagine aux prunes)

Algorithme : Recette Tagine aux prunes

Ingrédients : 1kg de viande de mouton, ½ kg de prunes, sel, …

Début

Cuire la viande

Ajouter les prunes et un peu de sel

Laisser cuire pendant une ½ heure

Servir chaud

Fin



Définition



Mais c'est quoi un algorithme ?

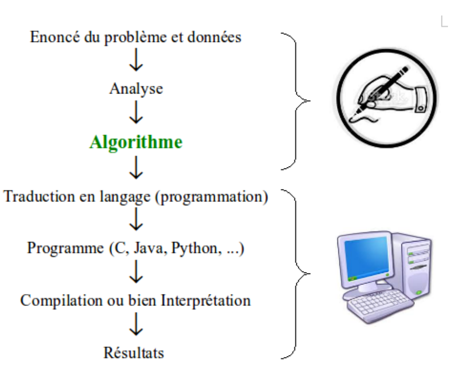
- **Définition 1** : Une description des différentes étapes permettant de résoudre un problème quelconque.
- **Définition 2** : Une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.

Définition



La résolution de problème et la programmation

La résolution d'un problème passe par ces étapes :

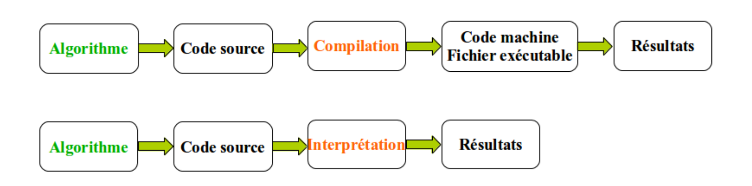


Définition



Compilation & Interprétation

Il existe deux modes de traduction d'un tel programme source en code binaire exécutable par la machine :



Définition



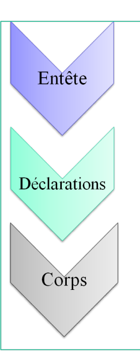
Compilation & Interprétation

- L'interpréteur prend une instruction, puis traduit et exécute puis prend une autre instruction. Alors que le compilateur traduit tout le programme en une seule fois (programme exécutable), puis l'exécute.
- Le compilateur génère le rapport d'erreur après la traduction de code source entier, alors qu'un interpréteur s'arrêtera la traduction tant qu'il obtient la première erreur.

Structure d'un algorithme



Un algorithme est composé de trois parties :



- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme + Nom

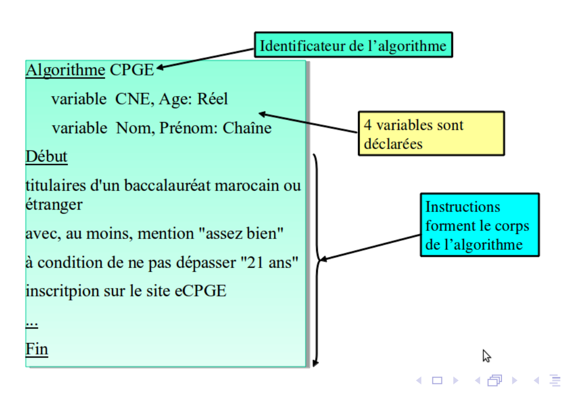
- Déclarations des données utilisées dans le corps de l'algorithme : variable, constante, fonction, …

- C'est dans cette partie que les tâches (instructions, opérations,…) de l'algorithme sont placées
- Mots-clés : Début, Fin

Structure d'un algorithme



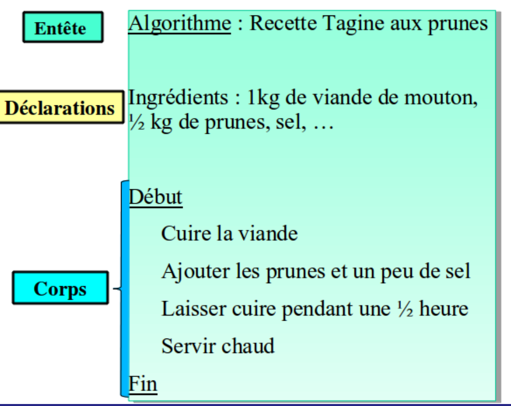
Algorithme d'accès aux CPGE



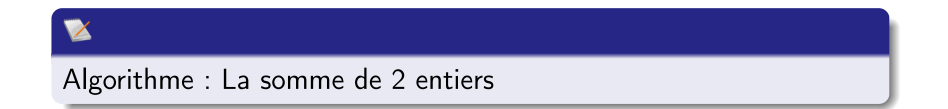
Structure d'un algorithme

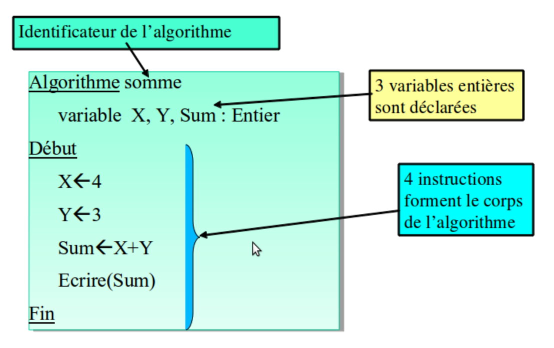


Algorithme de recette de cuisine : Tagine aux prunes



Structure d'un algorithme





Structure d'un algorithme



Entête

- L'entête d'un algorithme permet d'identifier (nommer) un algorithme
- **Exemple** : Si je décide d'écrire un algorithme qui réalise la somme de deux réels ayant comme nom **somme**, alors l'entête est :

Algorithme somme

Structure d'un algorithme



Déclaration

- La partie déclaration dans un algorithme contient une liste : des variables, des constantes,... utilisées dans le **corps de l'algorithme**
- Les **variables**, constantes,... utilisées doivent avoir fait l'objet d'une déclaration préalable
- Derrière toute **variable** on peut stocker une valeur.

Structure d'un algorithme



Variables

- Syntaxe : **nom_variable : type_variable**
- avec,
  - nom_variable : l'identifiant de la variable
  - type_variable : le type de la variable

Conseil : pour la lisibilité du code, choisissez des noms significatifs qui décrivent les données manipulées

Exemples : TotalVentes2014, Prix_TTC, Prix_HT

Structure d'un algorithme



Variables

Important : Le choix des noms de variables est soumis à quelques règles :

- Commencer par une lettre alphabétique
- Etre constitué uniquement de lettres, de chiffres et du soulignement "_" .
- Etre différent des mots réservés du langage

Déclaration valide : TotalVentes2014, Prix_TTC, Prix_HT **Déclaration invalide :** 2014_12, 2014TotalVentes, Prix TTC, Prix-TTC

Structure d'un algorithme



Types de variables

- **Entier :** pour caractériser des nombres entiers positifs et négatifs
- **Réel :** pour caractériser des nombres réels positifs et négatifs
- **Caractère :** 'A', 'b', '#', '@', ' ?', ... (lettres, signes de ponctuation, espaces, ...)
- **Chaîne :** Chaîne de caractères : "Bonjour", ...
- **Booléen :** Vrai ou faux (0 ou 1)
- **Constante :** Variable qui n'évolue pas (ne change pas de valeur).

Structure d'un algorithme



Types de variables

Exemple :

- Dans l'algorithme de calcul de la somme de deux entiers, la variable **Sum :** est utilisée pour stocker la somme de deux entiers
- Sa déclaration est donc :

variable Sum : Entier

Structure d'un algorithme



Constantes

Syntaxe :

Constante nom_constante ← valeur_constante : type_variable

Exemple :

Constante pi ← 3,14 : Réel

Structure d'un algorithme



Corps

Rôle du corps : Pour définir les tâches (instructions, opérations,...) que l'algorithme doit effectuer pour résoudre l'énoncé d'un problème.

Syntaxe :

```
Début
    instruction 1
    instruction 2
    ...............
    ...............
Fin
```

Exemple :

```
Début
    X←4
    Y←3
    Sum←X+Y
    Ecrire(X+Y)
Fin
```

Structure d'un algorithme



Instructions d'affectation

Rôle : mettre une première valeur dans une variable ou changer sa valeur courante.

Exemples :

- Note1 ← 15
- Note2 ← Note
- Moyenne ← (Note2*2 +Note1)/3
- Nom ← "Mohamed"

Structure d'un algorithme



Instructions d'affectation

Remarque : L'instruction $c \leftarrow a + b$ se comprend de la façon suivante :

- Prendre la valeur contenue dans la variable a
- Prendre la valeur contenue dans la variable b
- Additionner ces deux valeurs
- Mettre ce résultat dans la variable c
- Si c avait auparavant une valeur, cette dernière est perdue

Structure d'un algorithme



Instructions d'affectation

Remarque :

- L'affectation n'est pas commutative : A $\leftarrow$ B est différente de B $\leftarrow$ A
- L'affectation est différente d'une équation mathématique :
  1. A $\leftarrow$ A+1 a un sens
  2. A+1 $\leftarrow$ 2 n'est pas possible et n'est pas équivalente à A $\leftarrow$ 1

Structure d'un algorithme



Opération sur les variables : Priorité

- Pour les opérateurs arithmétiques, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :
  1. ^ : (puissance)
  2. * , / : (multiplication, division)
  3. % : (modulo)
  4. + , - : (addition, soustraction)
- **Exemple** : 1 + 3 * 2^2 = 13
- En cas de besoin on utilise les parenthèses pour indiquer les opérations à effectuer en priorité
  Exemple : (1 + 3) * 2^2 = 16

Structure d'un algorithme



Instructions d'affectation

Remarque : La valeur affecté doit être compatible avec le type de la variable utilisée :

Exemple :

Variables i, j, k : entier
Variables x, y : réel
Variable OK : booléen
Variables ch1, ch2 : chaîne de caractères

Affectations valides :

i ← 1	j ← i	k ← i+j
x ← 10.3	OK ← FAUX	ch1 ← "SMI"
ch2 ← ch1	x ← 4	x ← j

Affectations non valides :

i ← 10.3	OK ← "SMI"	j ← x

Structure d'un algorithme



Exercice

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables   A, B : entier

Début
   A ← 5
   B ← A+4
   A ← A+1
   B ← A-4
   A ← B
   B ← A
Fin
```

Structure d'un algorithme



Corrigé

Instruction	Valeur de A	Valeur de B
0 :	NULL	NULL
1 : A ← 5	5	NULL
2 : B ← A+4	5	9
3 : A ← A+1	6	9
4 : B ← A-4	6	2
5 : A ← B	2	2
6 : B ← A	2	2

Structure d'un algorithme



Instructions d'entrée/sortie

Rôle :

- Pour assurer la communication entre l'ordinateur et l'extérieur on utilise les instructions de **lecture** et d'**écriture**
- Pour afficher un résultat (du texte ou le contenu d'une variable) ou bien demander à l'utilisateur de saisir une information afin de la stocker dans une variable

Structure d'un algorithme



Instruction de sortie

Syntaxe :

- Ecrire (valeur)
- Ecrire (variable)
- Ecrire (expression)

Exemples :

- Ecrire (4)
- Ecrire (Note)
- Ecrire ("La moyenne=", (Note1+Note2)/2)

Remarque : Ecrire(Note) n'est pas la même chose que Ecrire("Note")

Structure d'un algorithme



Instruction d'entrée

Syntaxe :

- Lire (variable)

Exemple :

- **Variable** Note : Réel
  Début
  Lire (Note)
  ...
  Fin

Effet

- A la rencontre de cette instruction, l'ordinateur arrête l'exécution du programme et attend qu'on tape une valeur.
- On termine la saisie en appuyant sur la touche Entrée.
- La valeur qu'on a tapé est affectée à la variable.

Remarque : Lire(valeur) et Lire(expression) n'ont pas de sens

Structure d'un algorithme



Instruction d'entrée

Conseil :

- Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

Exemple :

- **Variable** Note : Réel
  Début
  Ecrire ("Entrer la valeur de la Note :")
  Lire (Note)
  ...
  Fin



Schéma alternatif ou conditionnel



La sélection (réduite)

Exemple : Ecrire un algorithme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est positif.

Schéma alternatif ou conditionnel



La sélection (réduite)

Rôle : Le schéma conditionnel permet d'exécuter des instructions quand une condition est vérifiée.

Syntaxe :

Si condition **Alors**
    instructions
    ...
Fin Si

Schéma alternatif ou conditionnel



La sélection (simple)

Exemple : Ecrire un algorithme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est positif ou négatif.

Schéma alternatif ou conditionnel



La sélection (simple)

Rôle : Le schéma conditionnel permet d'exécuter des instructions quand une condition est vérifiée. et éventuellement dans le cas ou la condition n'est pas vérifiée.

Syntaxe :

Si condition **Alors**
    instructions
    ...
Fin Si
Sinon
    instructions
    ...
Fin Sinon

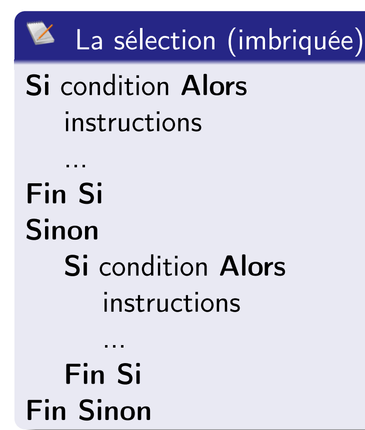
Schéma alternatif ou conditionnel



La sélection (imbriquée)

Exemple : Ecrire un algorithme qui vérifie si a, b et c sont ordonnées selon : $a \leq b \leq c$.

Schéma alternatif ou conditionnel



La sélection (imbriquée)

```
Si condition Alors
    instructions
    ...
Fin Si
Sinon
    Si condition Alors
        instructions
        ...
    Fin Si
Fin Sinon
```

Schéma itératif ou répétitif



Exemple :
Ecrire un algorithme qui affiche le mot "Bonjour" 100 fois.

Schéma itératif ou répétitif



Rôle

Un schéma itératif permet de répéter une même action un certain nombre de fois. On parle aussi de boucle.

Schéma itératif ou répétitif



Boucle : Pour

Principe :

On sait à l'écriture de l'algorithme combien on doit faire d'itérations.

Syntaxe :

Pour variable ← valeur_initiale **à** valeur_finale [**de pas** p] **Faire**

    instructions

    ...

Fin Pour

Schéma itératif ou répétitif



Exemple :

Ecrire un algorithme qui affiche le mot "Bonjour" 100 fois.

Solution :

```
Pour i ← 1 à 100 de pas 1 Faire
    Ecrire("Bonjour")
Fin Pour
```



Schéma itératif ou répétitif



Exemple :

Ecrire un algorithme qui affiche le mot "Bonjour" 100 fois.

Solution :

Pour i ← 1 à 100 **Faire**
    Ecrire("Bonjour")
Fin Pour



Schéma itératif ou répétitif



Exemple :

Ecrire un algorithme qui contrôle la saisie d'un nombre positif jusqu'à ce que le nombre entré soit valable.

Schéma itératif ou répétitif



Boucle : Tant que

Principe :

Dans cette boucle, on ne sait pas à l'avance le nombre d'itérations à faire. La sortie de la boucle s'effectue lorsque la condition n'est plus vérifiée.

Syntaxe :

Tant que condition **Faire**
    instructions
    ...
Fin Tant que



Schéma itératif ou répétitif



Exemple : Ecrire un algorithme qui contrôle la saisie d'un nombre positif jusqu'à ce que le nombre entré soit valable.

Solution :

```
Variable nbr : Réel
Debut
    Ecrire("Entrez un nombre")
    Lire(nbr)
    Tant Que (nbr < 0) Faire
        Ecrire ("Saisie erronée. Recommencez")
        Lire (nbr)
    Fin Tant Que
    Ecrire ("Saisie valable")
Fin
```

Tableaux à une dimension



Contexte :

- Moy = (N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 + N11 + N12) / 12
- Mais si on a un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, alors là comment faire ?

Tableaux à une dimension



Notion :

- Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans une seule structure de donnée appelée tableau.
- Un tableau est composé d'un certain nombre de cases :

8	3	5	5	7	4	9	1	3

Tableaux à une dimension



Syntaxe :

Variable nom_tab : **Tableau[Taille]** : Type

Exemple :

Variable notes : **Tableau[12]** : Réel

Tableaux à une dimension



Exemple : Initialisation d'un tableau par l'utilisateur.

```
Algorithme initialisation
Variable notes : Tableau[12] : Réel
Variable i : Entier
Debut
Pour i ← 1 à 12 Faire
    Ecrire ("Saisie la note ", i)
    Lire (notes[ i ])
Fin pour
Fin
```

Tableaux à une dimension



Exemple : Ecrire un algorithme qui calcule et affiche la moyenne des notes d'une classe de 36 étudiants.

```
Algorithme Moyenne
Variable notes : Tableau[36] : Réel
Variable SOM : Réel
Variable i : Entier
Debut
    SOM ← 0
    Pour i ← 1 à 36 Faire
        Ecrire ("Saisie la note ", i)
        Lire (notes[ i ])
        SOM ← SOM + notes[ i ]
    Fin pour
    Ecrire ("la moyenne des notes de la classe est :", SOM/36)
Fin
```

Tableaux à une dimension



Recherche séquentielle dans un tableau

Exemple : Ecrire un algorithme qui fait la recherche d'une valeur dans un tableau.

```
Algorithme Recherche_séquentielle
Variable Tab : Tableau[10] : Entier
Variables i,val,pos : Entier
Debut
   Pour i ← 1 à 10 Faire
      Ecrire ("Saisie l'élément", i)
      Lire (Tab[ i ])
   Fin pour
   Ecrire ("Saisie l'élément à rechercher :")
   Lire (val)
   pos ← -1
   Pour i ← 1 à 10 Faire
      Si val = Tab[ i ] Alors
         pos ← i
         break
      FinSi
   Fin pour
   Si pos = -1 Alors
      Ecrire ("La valeur recherchée ne se
      trouve pas")
   FinSi
   Sinon
      Ecrire ("La valeur ",val," se trouve
      dans la position ", pos)
   Fin Sinon
Fin
```

Tableaux à deux dimensions



Ceci est utile par exemple pour représenter des matrices

Syntaxe :

Variable nom_tab : **Tableau[Taille1][Taille2]** : Type

Exemple :

Variable matrice : **Tableau[3][4]** : Réel

Ceci va créer une variable matrice qui est un tableau contenant 3 lignes et 4 colonnes : **3*4**.

Tableaux à deux dimensions

Exemple :

Saisie des valeurs d'une matrice 5*4.

```
Algorithme SaisieMatrice
Variable matrice : Tableau[5][4] : Entier
Variables i,j : Entier
Debut
    Pour i ← 1 à 5 Faire
        Pour j ← 1 à 4 Faire
            Ecrire ("matrice[",i,"][",j,"] :")
            Lire(matrice[ i ][ j ])
        Fin pour
    Fin pour
Fin
```

Historique



- Apparu : en 1990
- Auteur : Guido van Rossum



Les types de données



- Le type **int** (Entier)
- le type **float** (Réel)
- Le type **bool** (Booléen)
- Le type **str** (Caractères)

Affectations



- Note1 = 15
- Note2 = Note
- Moyenne = (Note2*2 + Note1)/3
- Prenom = 'Mohamed'
- Mot = "L'enfant"

Affectation multiple : Note1, Note2 , Note3 = 15, 16, 14

Opération sur les variables

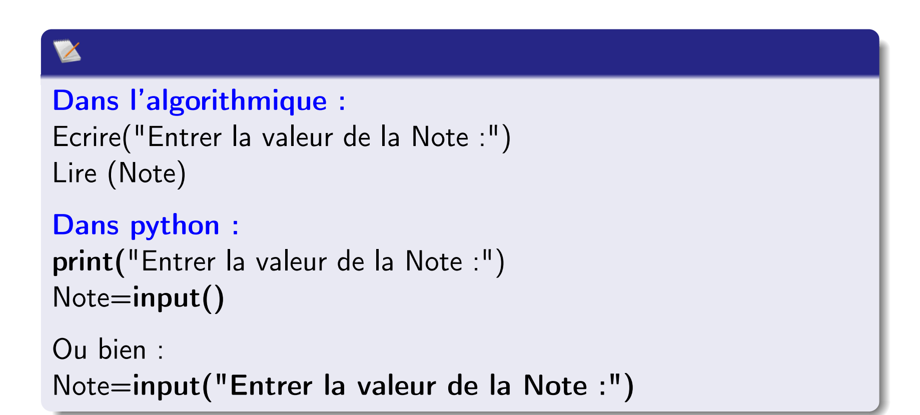
- ** : (puissance)
- * : (multiplication)
- / : (division)
- // : (division entière)
- % : (modulo)
- + , - : (addition, soustraction)
- = = : égal
- ! = Différent

Opération sur les variables



- x+=3 ⇔ x=x+3
- x-=3 ⇔ x=x-3
- x*=3 ⇔ x=x*3
- x%=3 ⇔ x=x%3

Les entrées/sorties standards



Dans l'algorithmique :

Ecrire("Entrer la valeur de la Note :")
Lire (Note)

Dans python :

print("Entrer la valeur de la Note :")
Note=**input()**

Ou bien :
Note=**input("Entrer la valeur de la Note :")**

Les entrées/sorties standards



Exemple

Ecrire un programme en python qui calcule la somme de deux Notes saisies au clavier

Schéma alternatif ou conditionnel



La sélection (réduite)

Rôle : Permet d'exécuter des instructions quand une condition est vérifiée.



Algorithme

Si condition **Alors**
    instructions
    ...
Fin Si



Python

```
if condition :
    instructions
    ...
```

Schéma alternatif ou conditionnel



La sélection (réduite)

Exemple : Ecrire un programme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est positif.

Schéma alternatif ou conditionnel



La sélection (simple)

Rôle : Permet d'exécuter des instructions quand une condition est vérifiée. et éventuellement dans le cas ou la condition n'est pas vérifiée.



Algorithme

Si condition **Alors**
    instructions
    ...
Fin Si
Sinon
    instructions
    ...
Fin Sinon

Python

```
if condition :
    instructions
    ...
else :
        instructions
        ...
```

Schéma alternatif ou conditionnel



La sélection (simple)

Exemple : Ecrire un programme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est positif ou négatif.

Schéma alternatif ou conditionnel



La sélection (imbriquée)



Algorithme

Si condition **Alors**
    Si condition **Alors**
        instructions
        ...
    Fin Si
Fin Si
Sinon
    Si condition **Alors**
        instructions
        ...
    Fin Si
Fin Sinon

Python

```
if condition :
    if condition :
        instructions
        ...
else :
    if condition :
        instructions
        ...
```

Schéma alternatif ou conditionnel



La sélection (imbriquée)

Exemple1 : Ecrire un programme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est positif ou négatif ou nul.

Exemple2 : Ecrire un programme qui vérifie si a, b et c sont ordonnées selon : $a \leq b \leq c$.

Schéma itératif ou répétitif



Boucle : Pour



Algorithme

Pour variable ← valeur_initiale **à** valeur_finale **de pas** p **Faire**
    instructions
    ...
Fin Pour



Python

```
for variable in range(valeur_initiale, valeur_finale, p) :
    instructions
    ...
```

Schéma itératif ou répétitif

Boucle : Pour

Exemples :

- ```
  for i in range(0, 10, 2) :
      print(i)
  ```
  donne le résultat : 0, 2, 4, 6, 8
- ```
  for i in range(0, 10) :
      print(i)
  ```
  donne le résultat : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- ```
  for i in range(10) :
      print(i)
  ```
  donne le résultat : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  ```

Schéma itératif ou répétitif



Boucle : Tant que



Algorithme

Tant que condition **Faire**
    instructions
    ...
Fin Tant que



Python

```
while condition :
    instructions
    ...
```



Exemple

```
i=0
while i<5 :
    print(i, end=' ')
    i+=1
```



Résultat

0 1 2 3 4



break



définition

Sort immédiatement de la boucle for ou while en cours contenant l'instruction

Exemple :

```
for i in range(10) :
    if i==5 :
        break
    print(i)
```

donne le résultat : 0, 1, 2, 3, 4

continue



définition

Sort immédiatement de la boucle **for** ou **while** en cours contenant l'instruction

Exemple :

```
for i in range(10) :
    if i==5 :
        continue
    print(i)
```

donne le résultat : 0, 1, 2, 3, 4, 6, 7, 8, 9

Les commentaires



définition

Un commentaire est du texte, qui accompagne le code source, et qui n'est pas pris en considération ni par l'interpréteur ni par le compilateur ; c'est comme si le commentaire n'existe plus dans le code source.
On l'utilise pour éclaircir le code, pour l'expliquer, pour donner une meilleure idée sur notre code source.
Pour écrire un commentaire on utilise le caractère #

Exemple :

```
i+=1        # c'est l'incrémentation de i par 1
```



Les fonctions



Contexte

Division du programme complexe en plusieurs sous-programmes :

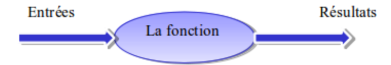
- Moins complexes
- Indépendants
- Réutilisables

Les fonctions

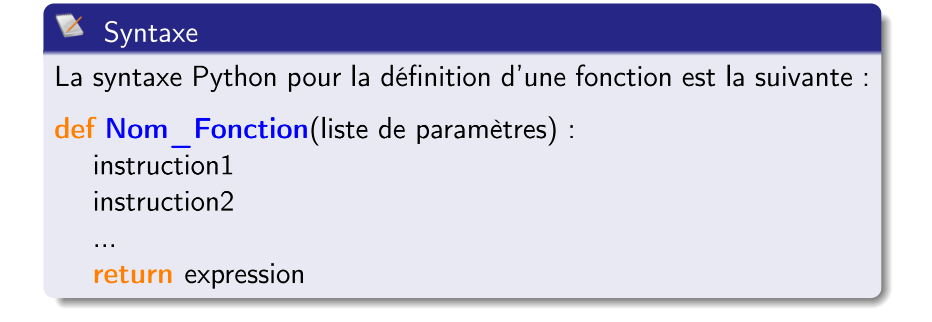
Principe

On connaît déjà certaines fonctions Python :

- print()
- input()
- sqrt()
- pow()
- ...



Les fonctions



Syntaxe

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def Nom_Fonction(liste de paramètres) :
    instruction1
    instruction2
    ...
    return expression
```

Les fonctions



Le retour de valeur(**return**)

- L'instruction **return** est une instruction qui se place à l'intérieur de la définition d'une fonction (elle peut ne pas exister, selon les cas)
- Elle indique que la fonction a terminé sa tâche et va renvoyer (ou retourner) un résultat.

Les fonctions



Syntaxe

NB :

Pour utiliser la fonction que nous venons de définir, il suffit de l'appeler par son nom. Ainsi :

Nom_Fonction(liste de paramètres)

Les fonctions



Exemple

- Ecrire une fonction **Multiple** qui affiche les 10 premiers multiples d'un entier a
- **5 10 15 20 25 30 35 40 45**
- Utiliser cette fonction dans un programme pour afficher les 10 premiers multiples des entiers : de 1 jusqu'à 9
- **1 2 3 4 5 6 7 8 9**
  2 4 6 8 10 12 14 16 18
  3 6 9 12 15 18 21 24 27
  4 8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81

Les fonctions



Exemple

- Reformuler la fonction **Multiple** pour afficher les n premiers multiples d'un entier a
- Utiliser cette fonction dans un programme pour afficher les n premiers multiples des entiers : de 1 jusqu'à 9

Les variables globales et locales



Les variables locales

- Les variables locales sont les variables qui n'existent que dans le corps de la fonction.
- Les paramètres sont aussi des variables locales.
- Quand une fonction est appelée, ses variables locales reçoivent les valeurs des paramètres d'appel.
- A la sortie de la fonction, les variables locales sont perdues.

Les variables globales et locales



Les variables globales

- Une variable globale définie en dehors de toute fonction.
- Une variable globale est connue dans toute la portion de code qui suit sa création.

Les variables globales et locales



Exemple : variable globale

```
x = 1
print(x)
```



Exemple : variable locale

```
def test() :
    y = 8
    return y
```

Les variables globales et locales

Exemple

Exécuter ce programme

```
def test() :
    x = 8
    print('x locale = ',x)
    return
x = 0
print('x globale = ',x)
test()
print('x globale = ',x)
```

Les variables globales et locales

Exemple

Exécuter ce programme

```
def test() :
    global x
    x = 8
    print('x locale = ',x)
    return
x = 0
print('x globale = ',x)
test()
print('x globale = ',x)
```

Les fonctions récursives



Définition

Une fonction récursive est une fonction qui s'appelle elle-même.

Exemple(le calcul de la factorielle) : $\forall n \geq 0, \; n! = \prod_{k=1}^{n} k$

```
def fact(n) :
    f=1
    for i in range(1,n+1) :
        f*=i
    return f
```

Les fonctions récursives



Définition

Une fonction récursive est une fonction qui s'appelle elle-même.

Exemple(le calcul de la factorielle) : $\forall n \geq 0,\ n! = \prod_{k=1}^{n} k$

```
def fact(n) :
    f=1
    for i in range(1,n+1) :
        f*=i
    return f
```

Les fonctions récursives



Définition

Une autre définition de la factorielle se fait par récurrence :

Exemple(le calcul de la factorielle) : $\forall n \geq 0,\ n! = \prod_{k=1}^{n} k$

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon.} \end{cases}$$

```
def fact_recursive(n) :
    if n==0 :
        return 1
    return n*fact_recursive(n-1)
```

Les fonctions récursives



Définition

Une autre définition de la factorielle se fait par récurrence :

Exemple(le calcul de la factorielle) : $\forall n \geq 0,\ n! = \prod_{k=1}^{n} k$

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon.} \end{cases}$$

```
def fact_recursive(n) :
    if n==0 :
        return 1
    return n*fact_recursive(n-1)
```

Les fonctions récursives



Définition

Une autre définition de la factorielle se fait par récurrence :

Exemple(le calcul de la factorielle) : $\forall n \geq 0,\ n! = \prod_{k=1}^{n} k$

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon.} \end{cases}$$

```
def fact_recursive(n) :
    if n==0 :
        return 1
    return n*fact_recursive(n-1)
```

Les fonctions récursives



Exercices

Exercice1(puissance) : Ecrire une fonction **puissance_recursive** prenant en entrée deux paramètres x et n et calculant récursivement x^n, en exploitant l'égalité : $x^n = x \times x^{n-1}$.

```
def puissance_recursive(x,n) :
  if n==0 :
    return 1
  return x*puissance_recursive(x,n-1)
```

Les fonctions récursives



Exercices

Exercice1(puissance) : Ecrire une fonction **puissance_recursive** prenant en entrée deux paramètres x et n et calculant récursivement x^n, en exploitant l'égalité : $x^n = x \times x^{n-1}$.

```
def puissance_recursive(x,n) :
    if n==0 :
        return 1
    return x*puissance_recursive(x,n-1)
```

Les fonctions récursives



Exercices

Exercice2(PGCD) : Ecrire l'algorithme d'Euclide sous la forme d'une fonction récursive

```
def pgcd_recursive(a,b) :
    r = a%b
    if r==0 :
        return b
    return pgcd_recursive(b,r)
```

Les fonctions récursives



Exercices

Exercice2(PGCD) : Ecrire l'algorithme d'Euclide sous la forme d'une fonction récursive

```
def pgcd_recursive(a,b) :
    r = a%b
    if r==0 :
        return b
    return pgcd_recursive(b,r)
```

Les chaînes de caractères



Notion

- Une chaîne de caractères est une suite finie de caractères consécutifs, qu'on note entre apostrophes ou guillemets :
  chaine1 = 'Tu sais programmer en python'
  chaine2 = '"Non", je crois que python est difficile !'
  chaine3 = "C'est facile"

Les chaînes de caractères



Notion

- A l'intérieur d'une chaîne de caractères, ***l'antislash*** (\) permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne, tabulation, apostrophes, guillemets, etc.)

  Exemple :

  \n dans une chaine provoque un saut à la ligne
  \t dans une chaine provoque une tabulation
- ***l'antislash*** (\) permet d'écrire sur plusieurs lignes une commande qui serait trop longue pour tenir sur une seule.

Les chaînes de caractères



Opération de concaténation

- La concaténation ne fonctionne que si tout les objets sont déjà de type chaînes de caractères
- **Exemple** :

```
ch1 = 'MPSI '
ch2 = 'et '
ch3 = 'PCSI'
ch4 = ch1 + ch2 + ch3
print(ch4)
MPSI et PCSI
```

Les chaînes de caractères



Opération de concaténation

- On ne peut pas additionner une chaîne et un nombre
- **Exemple** :

```
ch1 = 'MPSI'
ch2 = 1
ch3 = ch1 + ch2
TypeError : Can't convert 'int' object to str implicitly
```

Les chaînes de caractères



Opération de multiplication

- **Exemple** :

```
ch1 = 'HH !'
ch2 = ch1*4
print(ch2)
HH !HH !HH !HH !
```

Les chaînes de caractères



Accès à un caractère

- ch = 'abcdef'
- Pour accéder à chacun des caractères de la chaine de caractères on utilise ([i]) : où i est un indice qui indique la position du caractère

  Exemple :

  ```
  print(ch[3])
  'd'
  print(ch[-2])
  'e'
  ```

Les chaînes de caractères



Accès à un caractère

- ch = 'abcdef'
- On ne peut pas modifier les caractères d'une chaine de caractères :

  Exemple :

  ```
  ch = 'abcdef'
  ch[3]='D'
  TypeError : 'str' object does not support item assignment
  ```



Les chaînes de caractères



Longueur

- Pour déterminer la longueur (c'est-à-dire le nombre de caractères) d'une chaîne, en appel à la fonction **len**()

  Exemple :

```
ch = 'abcdef'
x = len(ch)
print(x)
6
```

Les chaînes de caractères



Sous-chaînes

ch = 'abcdefghij'

- Pour extraire une sous-chaîne de ch, on écrit ch[i :j] où i est l'indice du premier caractère de la sous-chaîne et j-1 est l'indice du dernier caractère :

  Exemple :

  ```
  print(ch[0 :5]) #ch[ :5]
  abcde
  print(ch[2 :5])
  cde
  ```

Les chaînes de caractères



Sous-chaînes

ch = 'abcdefghij'

NB :

- Si j ≤ i il n'y a pas de sous-chaîne correspondante. Python renvoie alors une chaîne vide :

```
print(ch[5 :2])
```

- Il faut préciser le pas :

```
print(ch[5 :2 :-1])
fed
```

Les chaînes de caractères



L'opérateur in

ch = 'abcdefghij'

l'opérateur in sert à :

- afficher les caractères d'une chaîne de caractères :

```
for c in ch :
    print(c, end=' ')
```

  a b c d e f g h i j

- tester l'appartenance d'un caractère à une chaîne de caractères :

```
if 'e' in ch :
    return True
```

Les tuples



Définition

- Python propose un autre type de données composites appelé **tuple**, comme une chaîne de caractères (c.à.d **n'est pas modifiable**), mais la différence, un tuple se compose des éléments de n'importe quel type
- Un tuple est une collection d'éléments placés entre parenthèses et séparés par des virgules

Les tuples



Exemple

- un tuple composé de 5 entiers :
  T1 = (1, 10, 5, 77, 9)
- un tuple composé d'un entier, un réel et un caractère :
  T2 = (1, 10.5, 'a')
- un tuple composé d'un entier, un réel et une chaîne de caractères :
  T3 = (1, 10.5, 'cpge')

Les tuples



Accès à un élément

- T = (1, 10.5, 4, 'cpge')
- Pour accéder à chacun des éléments d'un tuple on utilise ([i]) : où i est un indice qui indique la position de chaque élément

  Exemple :

  ```
  print(T[0])
  1
  print(T[-1])
  'cpge'
  ```

Les tuples



Concaténation

- T1 = (1, 10.5, 4, 'cpge')
- T2 = ('a', 'b', 6)
- Il est possible de coller un n-uplet et un p-uplet pour obtenir un (n + p)-uplet. On parle de concaténation. L'opérateur correspondant en Python est l'opérateur +

  Exemple :

  ```
  T3 = T1+T2
  print(T3)
  (1, 10.5, 4, 'cpge', 'a', 'b', 6)
  ```



Les tuples



L'opérateur in

T = (1, 10.5, 4, 'cpge')

l'opérateur in sert à :

- afficher les éléments d'un tuple :

```
for e in T :
    print(e, end=' ')
1 10.5 4 'cpge'
```

- tester l'appartenance d'un élément à un tuple :

```
if x in T :
```

Les tuples

La fonction len()

T = (1, 10.5, 4, 'cpge')

La fonction len(T) sert à déterminer la longueur de T, c'est-à-dire le nombre d'éléments de T

Exemple :

afficher les éléments d'un tuple :

```
for i in range(len(T)) :
    print(T[i], end=' ')
```

1 10.5 4 'cpge'

Les tuples

L'opération [i :j :k]

T = (1, 10.5, 4, 'cpge', 2, 5.8, 'a')

Pour extraire un sous-tuple **T_s** de **T**, on écrit T[**i** :**j** :**k**] où **i** est l'indice du premier élément de **T_s**, **j**-**1** est l'indice du dernier élément et **k** le pas

Exemple :

```
print(T[0 :3]) #T[ :3]
(1, 10.5, 4)
print(T[2 :5])
(4, 'cpge', 2)
print(T[5 :3 :-1])
(5.8, 2)
print(T[ : :-1])
('a', 5.8, 2, 'cpge', 4, 10.5, 1)
```

Les listes



Définition

- Une liste est un tuple dont on peut changer les valeurs des éléments
- Pour construire une liste, on remplace les parenthèses par des crochets

Les listes



Exemples

- une liste composée de 5 entiers :
  L1 = [1, 10, 5, 77, 9]
- une liste composée d'un entier, un réel et un caractère :
  L2 = [1, 10.5, 'a']
- une liste composée d'un entier, un réel et une chaîne de caractères :
  L3 = [1, 10.5, 'cpge']

Les listes



NB

- Il est possible de changer les éléments d'une liste :
- ```
  L = [1, 10, 5, 77, 9]
  L[3] += 3
  print(L)
  [1, 10, 5, 80, 9]
  ```
  ```

Les listes



Exemple

- L1 = [1, 10, 5, 77, 9]
- un tuple composé d'un entier, un caractère et une liste :

```
T = (5, 'a', L1)
print(T)
(5, 'a', [1, 10, 5, 77, 9])
```

- est ce qu'on peut modifier les éléments de T ?

Les listes



Exemple

- L1 = [1, 10, 5, 77, 9]
- T = (5, 'a', L1)
- T[0] = 3
  TypeError : 'tuple' object does not support item assignment
- T[2][0] = 3
  print(T)
  (5, 'a', [3, 10, 5, 77, 9])

Les listes



Remarques

Toutes les opérations vues pour les tuples et les chaînes de caractères sont définies :

- l'opérateur in
- la fonction len()
- L'opérateur des tranches : [début : fin : pas]

Les listes



Concaténation

Exemple :

```
L1 = [1, 10.5, 4, 'cpge']
L2 = ['a', 'b', 6]
L3 = L1 + L2
print(L3)
[1, 10.5, 4, 'cpge', 'a', 'b', 6]
```

Les listes



Exercice

- Ecrire un programme qui permet d'initialiser par le clavier et d'afficher une liste de n éléments
- La même chose pour un tuple de n éléments

Les listes



Exercice

- Ecrire un programme qui permet d'initialiser par le clavier et d'afficher une liste de n éléments
- La même chose pour un tuple de n éléments

liste

```
n = int(input('entrer n : '))
L = [ ]
for i in range(n) :
    L += [int(input())]
print(L)
```

tuple

```
n = int(input('entrer n : '))
T = ( )
for i in range(n) :
    T += (int(input()), )
print(T)
```

Les listes



Exercice

- Ecrire un programme qui permet d'initialiser par le clavier et d'afficher une liste de n éléments
- La même chose pour un tuple de n éléments

Les listes



Exercice

- Ecrire un programme qui permet d'initialiser par le clavier et d'afficher une liste de n éléments
- La même chose pour un tuple de n éléments

liste

```
n = int(input('entrer n : '))
L = [0]*n
for i in range(len(L)) :
    L[i] = int(input())
print(L)
```

tuple

```
n = int(input('entrer n : '))
T = ( )
for i in range(n) :
    x = int(input())
    T += (x, )
print(T)
```

Les ensembles (set)



Définition

- Un ensemble est une collection d'objets sans **répétition** et **sans ordre**
- Un ensemble est noté par des accolades {...}. Exemples :
  {1, 3, 7}   {'c', 'p', 'g', 'e'}   {'cpge',0,(1,2,3)}
- L'ensemble vide se note set() et non {}

Les ensembles (set)



Comment construire un ensemble ?

- En extension :
```
>>> E = {1, 3, 2, 7}
>>> E
{1, 2, 3, 7}
```
- En compréhension :
```
>>> E = {x*x for x in range(5)}
>>> E
{0, 1, 4, 9, 16}
>>> E = {x*x for x in range(5) if x%2 == 0}
>>> E
{0, 16, 4}
```
- Avec la fonction **set**() :
```
>>> E = set('cpge')
>>> E
{'c', 'g', 'e', 'p'}
>>> E = set([1, 2, 3, 1, 2, 35])
>>> E
{1, 2, 3, 35}
```

Les ensembles (set)



Opérations sur les ensembles

- En ajoutant un élément à un ensemble E avec la méthode E.add(x) :
  > > > E = {1, 3, 2, 7}
  > > > E.add(5)
  > > > E
  {1, 2, 3, 5, 7}
- L'**union** $E \cup F = \{x : x \in E \text{ ou } x \in F\}$ se note E | F en Python :
  > > > {3,2,5,4} | {1,7,2,5} #{3,2,5,4}.union({1,7,2,5})
  {1, 2, 3, 4, 5, 7}
- L'**intersection** $E \cap F = \{x : x \in E \text{ et } x \in F\}$ se note E & F en Python :
  > > > {3,2,5,4} & {1,7,2,5} #{3,2,5,4}.intersection({1,7,2,5})
  {2, 5}
- L'**différence** $E - F = \{x : x \in E \text{ et } x \notin F\}$ se note E - F en Python :
  > > > {3,2,5,4} - {1,7,2,5} #{3,2,5,4}.difference({1,7,2,5})
  {3, 4}
- Ne pas confondre E - x qui construit un nouvel ensemble, avec la méthode E.remove(x) qui supprime x de l'ensemble E.

Les ensembles (set)



Accès à un élément

- E = {1, 2, 3, 4}
- Les éléments d'un ensemble ne sont pas numérotés. On ne peut pas utiliser une notation comme E[i]
- Pour accéder aux éléments d'un ensemble on utilise l'opérateur **in**
- L'opérateur **in** permet de savoir si un objet appartient à un ensemble :
  > > > 2 in {1,2,3,4}
  True
  > > > 5 in {1,2,3,4}
  False
- L'opération E < F permet de tester si l'ensemble E est strictement inclus dans l'ensemble F :
  > > > {2,4} < {1,2,3,4}
  True

Les dictionnaires



Définition 1

- Les dictionnaires sont un autre type composite
- Ils ressemblent aux listes dans une certaine mesure (ils sont modifiables comme elles)
- Ils ne sont pas des séquences
- L'index d'un dictionnaire appellé une **clé**

Les dictionnaires



Définition 2

Un dictionnaire est un tableau associatif, c.à.d. un type de données permettant de stocker des couples {clé : **valeur**}, avec un accès à la valeur à partir de la clé, la clé ne pouvant être présente qu'une seule fois dans le dictionnaire.

Les dictionnaires



Création d'un dictionnaire

- Les éléments d'un dictionnaire sont enfermés dans une paire d'**accolades**
- Un dictionnaire vide sera donc noté { }

```
>>> D1 = {}
>>> D1['nom'] = 'El mohamadi'
>>> D1['prenom'] = 'Mohamed'
>>> D1['age'] = 22
>>> D1
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>> D2 = {'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>> D2
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>> D3 = dict(nom='El mohamadi',prenom='Mohamed',age=22)
>>> D3
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
```

Les dictionnaires



Création d'un dictionnaire

- Les éléments d'un dictionnaire sont enfermés dans une paire d'**accolades**
- Un dictionnaire vide sera donc noté { }

```
>>> D4 = dict((('nom','El mohamadi'),('prenom','Mohamed'),('age',22)))
>>> D4
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>> D5 = dict([['nom','El mohamadi'],['prenom','Mohamed'],['age',22]])
>>> D5
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>> D6 = dict([('nom','El mohamadi'),('prenom','Mohamed'),('age',22)])
>>> D6
{'nom': 'El mohamadi', 'age': 22, 'prenom': 'Mohamed'}
>>>
```

Les dictionnaires



Accès aux éléments

- Pour accéder aux éléments contenus dans le dictionnaire, on utilise des **clés**

```
>>> D = {'nom': 'El mohamadi', 'prenom': 'Mohamed', 1 :[11,8,2012]}
>>> D['prenom']
'Mohamed'
>>> D[1]
[11, 8, 2012]
>>> D[1][2]
2012
```

Les dictionnaires



Enlever des éléments d'un dictionnaire

- L'instruction del vous permet d'effacer des éléments d'un dictionnaire en fonction de leur clé

```
>>> D = {'nom': 'El mohamadi', 'prenom': 'Mohamed', 1 :[11,8,2012]}
>>> del(D['prenom'])
>>> D
{'nom': 'El mohamadi', 1: [11, 8, 2012]}
>>>
```

Les dictionnaires



Parcours d'un dictionnaire

- Vous pouvez utiliser une boucle **for** et l'opérateur **in**
- L'ordre dans lequel les éléments seront extraits est **imprévisible** (puisqu'un dictionnaire n'est pas une séquence)

```
>>> D = {'nom':'El mohamadi', 'prenom':'Mohamed',
         1:[11,8,2012]  ,  'ville':'Tanger'}
>>> for elt in D:
        print(elt)

ville
nom
prenom
1
```

```
>>> for elt in D:
        print(D[elt])

Tanger
El mohamadi
Mohamed
[11, 8, 2012]
```

Les dictionnaires



Parcours d'un dictionnaire

- Vous pouvez utiliser la méthode **keys()** pour afficher les clés
- Pour effectuer un traitement sur les valeurs, utiliser la méthode **values()**

```
>>> D = {'nom': 'El mohamadi', 'prenom': 'Mohamed', 1 :[11,8,2012], 'Ville'
:'Tanger'}
>>> for elt in D.keys():
        print(elt)


Ville
nom
prenom
1
```

```
>>> for elt in D.values():
        print(elt)


Tanger
El mohamadi
Mohamed
[11, 8, 2012]
```

Les dictionnaires



Exercice

1. Ecrire une fonction **Eleve(n)** qui reçoit en paramètre le nombre des élèves **n**, et qui retourne un dictionnaire **eleves**, dans lequel vous mémoriserez les noms des élèves, leur âge et leur taille,
   utilisez une boucle pour accepter les données entrées par l'utilisateur.
   Dans le dictionnaire, le nom de l'élève servira de clé d'accès, et les valeurs seront constituées de tuples (âge, taille), dans lesquels l'âge sera exprimé en années (donnée de type entier), et la taille en mètres (donnée de type réel).
2. Ecrire une fonction **Afficher(eleves, nom)** dans laquelle l'utilisateur pourra fournir un nom quelconque pour obtenir en retour le couple "âge, taille" correspondant.
   Le résultat devra être une ligne de texte, telle par exemple :
   "Nom : Mohamed Brahimi - âge : 15 ans - taille : 1.74 m"

Les dictionnaires

```
def Eleve(n):
    eleve={}
    for i in range(n):
        nom=input('Entrer nom : ')
        age=int(input('Entrer âge : '))
        taille=float(input('Entrer taille : '))
        eleve[nom]=(age,taille)
    return eleve

def Afficher1(eleve, nom):
    for cle in eleve:
        if nom == cle:
            print("Nom :",cle,"- âge :",eleve[cle][0],
                  "ans - taille :",eleve[cle][1],"m")
            return
    print("L'élève",nom,"n'existe pas")
    return

def Afficher2(eleve, nom):
    for cle,valeur in eleve.items():
        if nom == cle:
            print("Nom :",cle,"- âge :",valeur[0],
                  "ans - taille :",valeur[1],"m")
            return
    print("L'élève",nom,"n'existe pas")
    return
```

Les dictionnaires



Types de clés

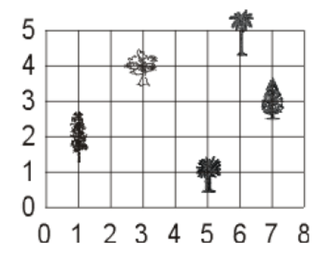
- Les clés ne sont pas nécessairement des chaînes de caractères ou bien des nombres
- Les clés sont de n'importe type non modifiable (c.à.d. de tout type sauf les types list et dict

Les dictionnaires

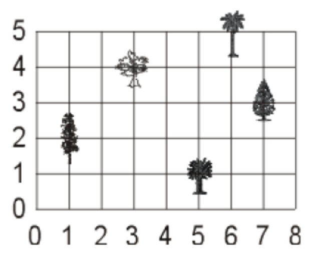


Types de clés

Exemple : Considérons par exemple que nous voulions répertorier des arbres situés dans un terrain rectangulaire. Nous pouvons pour cela utiliser un dictionnaire, dont les clés seront des tuples indiquant les coordonnées x,y de chaque arbre :



Les dictionnaires




```
>>> arbres = {}
>>> arbres[(1,2)] = 'Peuplier'
>>> arbres[(3,4)] = 'Platane'
>>> arbres[6,5] = 'Palmier'
>>> arbres[5,1] = 'Cycas'
>>> arbres[7,3] = 'Sapin'
>>> arbres
{(1, 2): 'Peuplier', (5, 1): 'Cycas', (3, 4): 'Platane', (6, 5): 'Palmier',
(7, 3): 'Sapin'}
>>>
```

Les fichiers textes



Pourquoi lire ou écrire dans des fichiers ?

- On rappelle que, l'ordinateur n'exécute que les programmes présents dans sa mémoire volatile (RAM)
- Quand vous fermez vos programmes, aucune de vos variables de ces programmes n'est sauvegardée
- Pour conserver durablement des informations, il faut utiliser une mémoire permanente comme par exemple le disque dur, la clé USB, le DVD,. . . sur lesquels le système d'exploitation organise les données sous la forme de **fichiers**

Les fichiers textes



Pourquoi lire ou écrire dans des fichiers ?

Les fichiers peuvent être, un excellent moyen :

- $\Rightarrow$ de produire des programmes qui utilisent ou produisent des données volumineuses
- $\Rightarrow$ de garder les valeurs de certains objets pour pouvoir les récupérer quand vous rouvrirez votre programme.

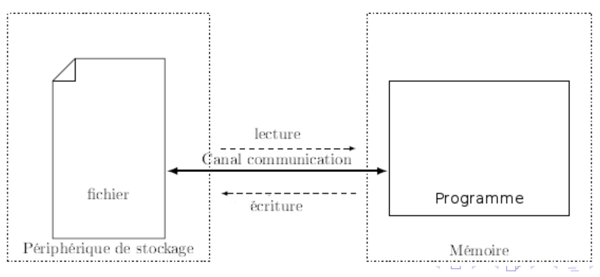
Les fichiers textes



Lecture et écriture dans des fichiers

Une lecture ou une écriture dans un fichier se fait en trois phases :

- On ouvre le fichier (on doit indiquer le nom du fichier)
- On lit ou on écrit dans le fichier
- On ferme le fichier



Les fichiers textes



Lecture à partir d'un fichier

En pratique :

- L'ouverture du fichier consiste à utiliser (la fonction **open**) à un fichier existant
- La lecture des lignes peut se faire soit :
  - globalement via la méthode **read**
  - pas à pas via la méthode **readline**
- Chaque ligne est en fait une chaîne de caractères (str)
- On ferme le fichier via la méthode **close**



Les fichiers textes



Lecture à partir d'un fichier

Exemple 1 :

La méthode **read()** renvoie tout le contenu du fichier, que l'on capture dans une chaîne de caractères contenu

```
#le fichier test.txt déjà existe
Fich = open('test.txt','r') # Ouverture du fichier text par open
                            # 'r' pour lecture
contenu = Fich.read() # La fonction read renvoie tout le contenu du fichier
print(contenu)

Fich.close () # Fermeture du fichier
```

Les fichiers textes



Lecture à partir d'un fichier

Exemple 2 :

- La méthode **read(n)** peut également être utilisée avec un argument. Celui-ci indiquera combien de caractères doivent être lus, à partir de la position déjà atteinte dans le fichier
- Si la fin du fichier est déjà atteinte, **read** renvoie une chaîne vide

```
>>> Fich = open('test.txt','r') # Ouverture du fichier par la méthode open
>>> chaine1=Fich.read(4)          # Lecture de 4 premiers caractères
>>> print(chaine1)
lign
>>> chaine2=Fich.read(2)#Lecture de 2 caractères à partir de la position
                         déjà atteinte dans le fichier
>>> print(chaine2)
e1
>>>
```

Les fichiers textes



Lecture à partir d'un fichier

Exemple 3 : En utilisant une boucle **for**
Affichage des lignes d'un fichier une à une

```
Fich = open('test.txt','r') # Ouverture du fichier par la méthode open
                            # 'r' pour la lecture

for elt in Fich:

    print(elt)   # affichage ligne par ligne

Fich.close()   # fermeture du fichier
```

Les fichiers textes



Lecture à partir d'un fichier

Exemple 4 :

- La méthode **readline()**, ne lit qu'une seule ligne à la fois
- La méthode **readlines()** transfère toutes les lignes dans une liste de chaînes
- À la fin du fichier, **readline** renvoie une chaîne vide, tandis que **readlines** renvoie une liste vide

```
Fich = open('test.txt','r') # Ouverture du fichier par la méthode open
                            # 'r' pour la lecture

print(Fich.readline()) # lecture et affichage de la première ligne

Liste = Fich.readlines() # transfère toutes les lignes restantes
                         # dans une liste de chaînes

for elt in Liste: # ou bien for elt in Fich ou for elt in Fich.readlines()
    print(elt)    #affichage les lignes restantes

Fich.close()   # fermeture du fichier
```

Les fichiers textes



Ecriture dans un fichier

Voici les principaux modes : **w** ou **a** :

- Fich1 = **open**('test.txt', 'w') : ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est crée ;
- Fich2 = **open**('test.txt', 'a') : ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est crée.

Les fichiers textes



Ecriture dans un fichier

Exemple 5 :

La méthode **write()** n'accepte en paramètre que des chaînes de caractères. Si vous voulez écrire dans votre fichier des nombres, il vous faudra les convertir en chaîne avant de les écrire et les convertir en entier après les avoir lus

```
>>> Fich = open('test.txt','w') # Ouverture du fichier en mode w
>>> ch = 'cpge'
>>> Fich.write(ch) # écrit la chaîne ch dans Fich
4
>>> Fich.close()
>>> Fich = open('test.txt','a') # Ouverture du fichier en mode a
>>> Liste = ['pcsi','mpsi']
>>> Fich.writelines(Liste) # écrit les chaînes de la liste Liste dans Fich
>>> Fich.close()
>>> Fich = open('test.txt','r') # Ouverture du fichier en mode r
>>> print(Fich.read())
cpgepcsimpsi
>>>
```

Les fichiers textes



Ecriture dans un fichier

Exemple 6 :

Ou bien on peut utiliser une deuxième option avec le paramètre **file=FICH** de la fonction **print**

```
>>> Fich = open('test.txt','w')
>>> print("cpge", file=Fich) # utilisation de l'option file
>>> Fich.close()
>>>
```

Les fichiers textes



Ecriture dans un fichier

Exercice :

Vous avez à votre disposition un fichier texte **f1.txt** dont chaque ligne est la représentation d'une valeur numérique de type réel. Par exemple :

1.8
2.6
3.2

Ecrire une fonction qui recopie ces valeurs dans un autre fichier **f2.txt** en les arrondissant en nombres entiers

Les Exceptions



Introduction

- Afin de rendre les codes python plus efficaces, il est recommandé de gérer les erreurs d'exécution des parties sensibles du code.
- La gestion des exceptions sépare d'un côté les instructions à exécuter lorsque tout se passe bien et, d'un autre côté, une ou plusieurs séquences d'instructions à exécuter en cas d'erreur.

Les Exceptions



Exemple

Soit le code suivant qui demande de saisir un nombre, puis il calcule et affiche son inverse :

```
nombre = float(input('Entrer un nombre : '))
inverse = 1/nombre
print("L'inverse de", nombre, "est :", inverse)
```

Les Exceptions



Exemple

Si vous entrez un nombre, tout se passe bien :

Entrer un nombre : 10

L'inverse de 10.0 est : 0.1



Exception

Mais que se passe-t-il autrement ?

Entrer un nombre : bonjour

Traceback (most recent call last) :

File "<pyshell#34>", line 1, in <module>

nombre = float(input('Entrer un nombre : '))

ValueError : could not convert string to float : 'bonjour'

Les Exceptions



Exception

Mais que se passe-t-il autrement ?

Entrer un nombre : 0

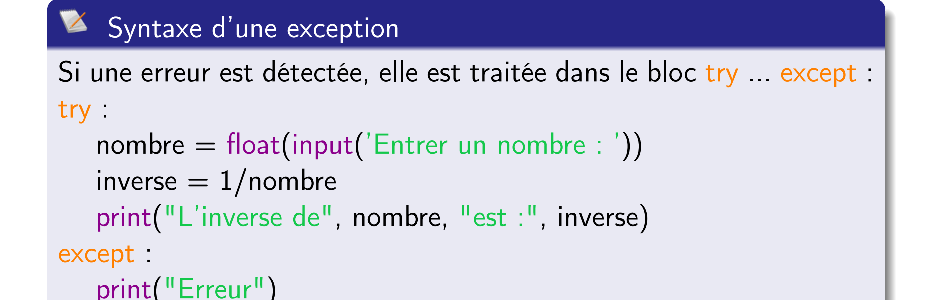
Traceback (most recent call last) :

File "<pyshell#36>", line 1, in <module>

inverse = 1/nombre

ZeroDivisionError : float division by zero

Les Exceptions



Syntaxe d'une exception

Si une erreur est détectée, elle est traitée dans le bloc try ... except :

```
try :
    nombre = float(input('Entrer un nombre : '))
    inverse = 1/nombre
    print("L'inverse de", nombre, "est :", inverse)
except :
    print("Erreur")
```

Les Exceptions



Syntaxe d'une exception

On peut distinguer les différents types d'exceptions :

```
try :
    nombre = float(input('Entrer un nombre : '))
    inverse = 1/nombre
    print("L'inverse de", nombre, "est :", inverse)
except ValueError :
    print(nombre, "n'est pas un nombre")
except ZeroDivisionError :
    print("Division par zéro")
```

Les Exceptions

Syntaxe complète d'une exception

```
try :
    # séquence normale d'exécution
except <exception_1> as e1 :
    # traitement de l'exception 1
except <exception_2> as e2 :
    # traitement de l'exception 2

else :
    # clause exécutée en l'absence d'erreur
finally :
    # clause toujours exécutée
```

Les Exceptions



L'instruction raise

L'instruction raise permet de lever volontairement une exception :

Exemple :

```
def factorial(n) :
    if n < 0 :
        raise ValueError("n doit être >= 0")
    if n%2 == 0 :
        raise ValueError("n doit être un entier")
    f = 1
    i = 2
    while i <= n :
        f *= i
        i += 1
    return f
```

Les Exceptions



Exercices

1. Modifier le code python précédent de manière à ressaisir le nombre en cas d'erreur.
2. Ecrire une fonction **racine**(nbr) qui calcule la racine carrée d'un nombre **nbr**, avec gestion des exceptions.
3. Ecrire une fonction **lireFichier**(nomFichier) qui lit le contenu du fichier **nomFichier**, avec gestion des exceptions.

POO



Introduction

Type de programmation :

1. Programmation impérative :
   - séquences d'instructions
   - l'affectation des variables
   - l'appel à des fonctions
   - les conditions (if ... else ... elif ...)
   - les boucles (for et while)
2. Programmation fonctionnelle : emboîtement de fonctions
3. Programmation objet : organisation d'un programme en le groupant en objets

POO



Introduction

Exemple : Classe : Personne = Attributs + Méthodes

1. Attributs : Nom, Prenom, Age, ...
2. Méthodes : Manger(), Boire(), ...

Personne
Non Prenom ⋮
def Manger() ⋮

3. Classe : Ensemble incluant des variables (attributs) et des fonctions (méthodes)

POO



Méthodes

- Une méthode décrit le comportement des objets instanciés à partir d'une classe
- Une méthode c'est une fonction avec un premier paramètre **self** (**obligatoire**)

```
class Personne :
    ⋮
    def Manger(self) :
        ⋮
```

POO

Exemple complet

- Classe Personne

```
class Personne :
    def __init__(self,nom,prenom,age) :#Constructeur
        self.nom = nom #nom est l'attribut d'instance
        self.prenom = prenom #prenom est l'attribut d'instance
        self.age = age #age est l'attribut d'instance

    def Manger(self) :#Méthode 1
        print(self.nom+" mange")

    def Boire(self) :#Méthode 2
        print(self.nom+" boit")
```

POO



Instanciation

- Les classes sont des fabriques d'objets : on construit d'abord l'usine avant de produire des objets (instances)
- On instancie un objet (c-à-d. qu'on le produit à partir de l'usine (classe)) en appelant le nom de sa classe comme s'il s'agissait d'une fonction

```
class Personne :
    def __init__(self,nom,prenom,age) :#Constructeur
        self.nom = nom #nom est l'attribut d'instance
        self.prenom = prenom #prenom est l'attribut d'instance
        self.age = age #age est l'attribut d'instance
    def Manger(self) :#Méthode 1
        ...
p1 = Personne("FAHSI","Ali",30) #p1 est une instance de la classe Personne
print(p1.age)
30
```

POO



Initialisateur ou bien Constructeur

- Un constructeur : est une méthode (fonction), sans return
- Porte un nom imposé par Python : __init__
- Lors de l'instanciation d'un objet, le constructeur __init__ est automatiquement appelé pour initialiser l'objet.

```
class Point :
    def __init__(self,x,y) :#Constructeur
        self.x = x #x est l'attribut d'instance
        self.y = y #y est l'attribut d'instance

pt1 = Point(1,4) #instanciation de l'objet pt1
print(pt1.x)
1
print(pt1.y)
4
```

POO

```
class Point :
    def __init__(self,x,y) :#Constructeur
        self.x = x #x est l'attribut d'instance
        self.y = y #y est l'attribut d'instance
    def afficherX(self) :#Méthode 1
        return self.x
    def afficherY(self) :#Méthode 2
        return self.y
    def deplacerPoint(self,dx,dy) :#Méthode 3
        self.x+=dx
        self.y+=dy
pt1 = Point(1,4) #instanciation de l'objet pt1
print(pt1.afficherX(), pt1.afficherY())
1, 4
pt1.deplacerPoint(2,3)
print(pt1.x, pt1.y)
3, 7
```

POO



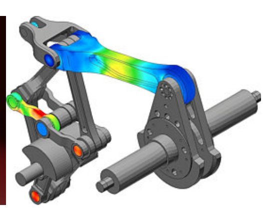
Exercice1

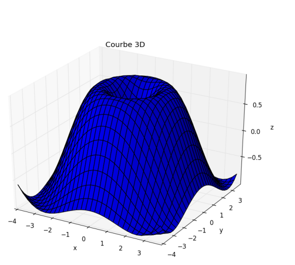
Ecrire une classe Employe avec les attributs suivants :

- nom : Le nom de famille de l'employé,
- prenom : Le prénom de l'employé.
- salaire : Le salaire mensuel
- La classe Employe doit disposer du constructeur suivant : __init__(self, nom, prenom, salaire)
- La classe Employe doit contenir les méthodes suivantes : information(self) : affiche les information d'un employé, gains(self) : retourne le salaire.

Ingénierie numérique et simulation







Ingénierie numérique et simulation



Problématique

⇒ Souvent, il n'y a pas de **solutions analytiques** pour des formules mathématiques.



Solution

- Le calcul scientifique sert à résoudre des problèmes rencontrés en mathématiques, sciences physiques ou chimie grâce à l'utilisation d'**algorithmes** numériques performants.
- On s'interesse pas à résoudre la formule mathématique donnée.

  ⇒Mais on s'interesse à une solution numérique aussi proches que possible de la solution réelle.

Ingénierie numérique et simulation



Problématique

⇒ Souvent, il n'y a pas de **solutions analytiques** pour des formules mathématiques.



Solution

- Le calcul scientifique sert à résoudre des problèmes rencontrés en mathématiques, sciences physiques ou chimie grâce à l'utilisation d'**algorithmes** numériques performants.
- On s'interesse pas à résoudre la formule mathématique donnée.

  ⇒Mais on s'interesse à une solution numérique aussi proches que possible de la solution réelle.

Ingénierie numérique et simulation



Problématique

⇒ Souvent, il n'y a pas de **solutions analytiques** pour des formules mathématiques.



Solution

- Le calcul scientifique sert à résoudre des problèmes rencontrés en mathématiques, sciences physiques ou chimie grâce à l'utilisation d'**algorithmes** numériques performants.
- On s'interesse pas à résoudre la formule mathématique donnée.

  ⇒Mais on s'interesse à une solution numérique aussi proches que possible de la solution réelle.

Ingénierie numérique et simulation



Introduction

- Ce chapitre explique comment utiliser les fonctions fournies par Python et ses bibliothèques (**numpy**, **scipy** et **matplotlib**)
  - Module **numpy** : traitement des tableaux
  - Module **scipy** : méthodes numériques (Résolution d'équations sur les réels ($f(x) = 0$), d'équations differentielles, ...)
  - Module **matplotlib** : représentation graphique

numpy



Les tableaux de **numpy**

- **numpy** permet de créer un tableau (**array**) de taille fixée, multidimensionnel contenant des élements de méme type et de même taille
- Pour utiliser **numpy** :
  1. import **numpy**
  2. Ou sous une abréviation : import **numpy** as **np**
  3. ou par la commande : from **numpy** import *
- **La syntaxe** : array(liste)

numpy



Les tableaux de **numpy**

- **numpy** permet de créer un tableau (**array**) de taille fixée, multidimensionnel contenant des élements de méme type et de même taille
- Pour utiliser **numpy** :
  1. import **numpy**
  2. Ou sous une abréviation : import **numpy** as **np**
  3. ou par la commande : from **numpy** import *
- **La syntaxe** : array(liste)

numpy



Les tableaux de **numpy**

- **numpy** permet de créer un tableau (**array**) de taille fixée, multidimensionnel contenant des élements de méme type et de même taille
- Pour utiliser **numpy** :
  1. import **numpy**
  2. Ou sous une abréviation : import **numpy** as **np**
  3. ou par la commande : from **numpy** import *
- **La syntaxe** : array(liste)

numpy



Les tableaux de **numpy**

- **numpy** permet de créer un tableau (**array**) de taille fixée, multidimensionnel contenant des élements de méme type et de même taille
- Pour utiliser **numpy** :
  1. import **numpy**
  2. Ou sous une abréviation : import **numpy** as **np**
  3. ou par la commande : from **numpy** import *
- **La syntaxe** : array(liste)

```
>>> import numpy as np
>>> #son contenu sous forme d'un tableau d'une dimension :
>>> a = np.array([3, 6, 2, 8, -3, -5, 9])
>>> a
array([ 3,  6,  2,  8, -3, -5,  9])
>>> #son contenu sous forme d'une matrice :
>>> m = np.array([[1,2,3],[4,5,6]])
>>> m
array([[1, 2, 3],
       [4, 5, 6]])
```

numpy



Création de tableaux

Indiquer le type des éléments du tableau avec l'instruction **dtype** :

- **dtype** = int pour les nombres entiers
- **dtype** = float pour les nombres réels
- **dtype** = complex pour les nombres complexes

```
>>> a1 = np.array([1.8,2.7,3],dtype=int)
>>> a1
array([1, 2, 3])
>>> a2 = np.array([1.8,2.7,3],dtype=float)
>>> a2
array([ 1.8,  2.7,  3. ])
>>> a3 = np.array([1.8,2.7,3],dtype=complex)
>>> a3
array([ 1.8+0.j,  2.7+0.j,  3.0+0.j])
>>> a1.dtype #pour connaître le type
dtype('int32')
```

numpy



Création de tableaux spéciaux

- La fonction **ones** pour créer des tableaux remplis par 1
- La fonction **zeros** pour créer des tableaux remplis par 0

```
>>> #pour créer un tableau de 2 colonnes et 3 lignes
     rempli par 1 de type réel
>>> a1 = np.ones((2,3),float)
>>> a1
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> #pour créer un tableau de 6 cases rempli par 0 de type entier
>>> a2 = np.zeros(6,int)
>>> a2
array([0, 0, 0, 0, 0, 0])
```

numpy



Création de tableaux spéciaux

- La fonction **arange** qui est l'équivalent de **range** avec un pas de type **float**
- La fonction **linspace** retourne un tableau dont ces éléments sont répartis entre deux bornes avec un nombre donné d'éléments

```
>>> #pour créer un tableau qui commençe par premier=3 et se termine
     par dernier=5(exlus) avec la possibilité d'avoir un pas de type réel
>>> a3 = np.arange(3,5,step=0.5)
>>> a3
array([ 3. ,  3.5,  4. ,  4.5])
>>> #pour créer un tableau de 10 éléments entre deux bornes 1 et 6
>>> a4 = np.linspace(1,6,10)
>>> a4
array([ 1.        ,  1.55555556,  2.11111111,  2.66666667,  3.22222222,
        3.77777778,  4.33333333,  4.88888889,  5.44444444,  6.        ])
```

numpy



Création de tableaux spéciaux

- Les fonctions **identity** et **eye** retournent la matrice identité (des 1 sur la diagonale et des zéros ailleurs)

```
>>> a1=np.identity(3) #matrice identité de taille 3x3
>>> a1
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> a2=np.eye(3) #matrice identité de taille 3x3
>>> a2
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> #matrice identité de taille 3x3 avec k=1 l'indice de la diagonale
>>> a3=np.eye(3, k=1)
>>> a3
array([[ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  0.]])
```

numpy



Informations d'un tableau

- **shape** retourne un tuple des dimensions d'un tableau
- **size** retourne La taille totale d'un tableau

```
>>> a = np.array([[1, 2, 3,], [4, 5, 6]])
>>> a.shape #pour savoir les dimensions de la matrice a
(2, 3)
>>> a.size  #pour savoir le nombre des éléments de la matrice a
6
```

numpy



Informations d'un tableau

Les fonctions : **sum( )**, **prod( )**, **mean( )**, **min( )** et **max( )**

```
>>> a = np.array([[1, 2, 3,], [4, 5, 6]])
>>> a.sum() #pour calculer la somme des éléments de la matrice a
21
>>> a.prod() #pour calculer le produit des éléments de la matrice a
720
>>> a.mean() #pour calculer la moyenne des éléments de la matrice a
3.5
>>> a.min() #pour retourner le min de a
1
>>> a.max() #pour retourner le max de a
6
>>>
```

numpy



Accès et modification d'éléments de tableaux

- La fonction **fill**(**v**) permet de remplir le tableau avec la valeur **v**

```
>>> a = np.array([[0, 1], [2, 3]]);a
array([[0, 1],
       [2, 3]])
>>> a.fill(5) # remplir le tableau par 5
>>> a
array([[5, 5],
       [5, 5]])
```

numpy



Opérations sur les tableaux

- Opérateur + : pour calculer la somme
- Opérateur - : pour calculer la soustraction
- Opérateur * : pour calculer ???

```
>>> a = np.array([[1, 2], [3, 4]]);a
array([[1, 2],
       [3, 4]])
>>> a + 2
array([[3, 4],
       [5, 6]], dtype=int32)
>>> a - 2
array([[-1,  0],
       [ 1,  2]], dtype=int32)
>>> a * 2
array([[2, 4],
       [6, 8]], dtype=int32)
```

numpy



Opérations sur les tableaux

- Opérateur + : pour calculer la somme
- Opérateur - : pour calculer la soustraction
- Opérateur ***** : **ATTENTION, ce n'est pas le produit matriciel**, c'est **le produit coefficient par coefficient**

```
>>> a = np.array([[1, 2], [3, 4]]);a
array([[1, 2],
       [3, 4]])
>>> b = np.array([[1, 2], [3, 4]]);b
array([[1, 2],
       [3, 4]])
>>> a + b
array([[2, 4],
       [6, 8]], dtype=int32)
>>> a - b
array([[0, 0],
       [0, 0]], dtype=int32)
>>> a * b
array([[ 1,  4],
       [ 9, 16]], dtype=int32)
```

numpy



Produit de matrices

La fonction **dot( )** retourne le produit de deux matrices (**Produit matriciel**)

```
>>> a = np.array([[1, 2], [3, 4]]);a
array([[1, 2],
       [3, 4]])
>>> b = np.array([[1, 2], [3, 4]]);b
array([[1, 2],
       [3, 4]])
>>> np.dot(a,b)# produit de a et b
array([[ 7, 10],
       [15, 22]])
```

numpy



Transposition de matrices

La fonction **transpose**( ) retourne la transposée d'une matrice

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]]);a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.transpose()
array([[1, 4],
       [2, 5],
       [3, 6]])
```

numpy



Transformation de matrices

La fonction **reshape(l,c)** retourne une nouvelle forme d'un tableau sans modifier ses données, avec **l** représente le nouveau nombre de lignes et **c** le nouveau nombre de colonnes

```
>>> a = np.array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]]);a
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
>>> a.reshape(6,2) #retourne une matrice de taille 6x2
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```

numpy



Aplatir une matrice

La fonction **flatten**( ) vue d'une dimension d'un tableau sans modification

```
>>> b = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]]);b
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
>>> b.flatten()
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

numpy



Copie ou référence

Les variables sont des références aux objets

```
>>> a = np.array([[0, 1], [2, 3]]);a
array([[0, 1],
       [2, 3]])
>>> b = a;b
array([[0, 1],
       [2, 3]])
>>> b[0,0] = 4;b # modifier l'élément b[0,0] par 4
array([[4, 1],
       [2, 3]])
>>> a # aussi l'élément a[0,0] est changé
array([[4, 1],
       [2, 3]])
```

numpy



Copie ou référence

La fonction **copy**() retourne une copie d'un tableau

```
>>> a = np.array([[0, 1], [2, 3]]);a
array([[0, 1],
       [2, 3]])
>>> b = a.copy() # la matrice b est une copie de la matrice a
>>> b
array([[0, 1],
       [2, 3]])
>>> b[0,0] = 4;b # modifier l'élément b[0,0] par 4
array([[4, 1],
       [2, 3]])
>>> a # l'élément a[0,0] n'est pas changé
array([[0, 1],
       [2, 3]])
```

numpy



Accès et modification d'éléments de tableaux

- a[i][j] ou bien a[i,j] correspond à l'élément de la ligne i et de la colonne j dans la matrice a

```
>>> a = np.array([[0, 1], [2, 3]]);a
array([[0, 1],
       [2, 3]])
>>> a[0,0] = 4 # modifier l'élément a[0,0] par 4
>>> a
array([[4, 1],
       [2, 3]])
```

numpy



Extraction des éléments d'un tableau

- Entre [ ] peut figurer une condition
- a[i] pour extraire la ligne i de la matrice a
- a[:,j] pour extraire la colonne j de la matrice a

```
>>> a = np.array([[1,2,3],[4,5,6]]);a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a[a>2] # retourne un tableau dont les éléments sont supérieurs à 2
array([3, 4, 5, 6])
>>> a[1] # retourne la ligne 1 sous forme d'un tableau
array([4, 5, 6])
>>> a[:,2] # retourne la colonne 2 sous forme d'un tableau
array([3, 6])
```

numpy



Extraction des sous-tableaux

$a[k : l, m : n]$ renvoie la sous-tableaux (a_{ij}) avec $k \leq i < l$ et $m \leq j < n$

```
>>> #matrice de taille 4x4 avec une diagonale : 1,2,3,4 et 0 ailleurs
>>> a = np.diag([1, 2, 3, 4]);a
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
>>> #Extraction du sous-tableau de la ligne 0 jusqu'à la ligne 3
>>> #et de la colonne 1 jusqu'à la dernière colonne
>>> a[0:3,1:]
array([[0, 0, 0],
       [2, 0, 0],
       [0, 3, 0]])
```

numpy



Valeurs aléatoires

- La fonction **np.random.rand** pour générer des valeurs aléatoires entre 0 et 1. (Pour tester des algorithmes matriciels).
- La fonction **np.random.random_integers**(i, j, k) retourne un vecteur de k nombres entiers entre i et j, j inclus

```
>>> a = np.random.rand(2,2);a#matrice 2x2 (coefficients aléatoires entre 0 et 1)
array([[ 0.12422618,  0.16630157],
       [ 0.42000401,  0.36946734]])
>>> a = np.random.random_integers(1,10,5);a
array([5, 3, 5, 6, 8])
```

numpy



Algèbre linéaire

- La fonction **det( )** permet de calculer le déterminant d'une matrice
- La fonction **inv( )** permet de calculer l'inverse d'une matrice

```
>>> a = np.array([[0,1],[2,3]]);a
array([[0, 1],
       [2, 3]])
>>> np.linalg.det(a)#pour calculer le déterminant de a
-2.0
>>> b = np.linalg.inv(a)# b=la matrice inverse de a
>>> b
array([[-1.5,  0.5],
       [ 1. ,  0. ]])
>>> i = b.dot(a)# i est la matrice identité
>>> i
array([[ 1.,  0.],
       [ 0.,  1.]])
```

numpy



Algèbre linéaire

- La fonction **eigvals( )** permet de calculer les valeurs propres
- La fonction **eig( )** permet de calculer les valeurs propres et base de vecteurs propres

```
>>> a = np.array([[0,1],[2,3]]);a
array([[0, 1],
       [2, 3]])
>>> np.linalg.eigvals(a)#valeurs propres
array([-0.56155281,  3.56155281])
>>> np.linalg.eig(a)#valeurs propres et base de vecteurs propres
(array([-0.56155281,  3.56155281]), array([[-0.87192821, -0.27032301],
       [ 0.48963374, -0.96276969]]))
>>>
```

numpy



Exercices

1. Créer un vecteur nul de la taille 10
2. Créer un vecteur nul de la taille 10 dont la case 5 est 1
3. Créer un vecteur avec des valeurs allant de 5 à 15
4. Créer un vecteur à 100 composantes commençant à 10 avec un pas de $\sqrt{2}$

matplotlib



matplotlib.pyplot

Pour tracer des graphes de fonctions nous n'utiliserons le sous-module **pyplot**, importé, avec alias, à l'aide de la commande :

```
import matplotlib.pyplot as plt
```

Les fonctions essentielles de **pyplot** sont :

- **plot**( ) : pour tracer des points et des courbes
- **show**( ) : pour afficher le graphique

matplotlib



matplotlib.pyplot

Utiliser la fonction **plot**(argument1, argument2, argument3) avec :

1. en 1er argument la liste des abscisses
2. en 2eme argument la liste des ordonnées
3. en 3eme argument [optionnel] le motif des points :
    - **'.'** : un petit point,
    - **'o'** : un gros point,
    - **'+'** : une croix,
    - **'*'** : une étoile,
    - **'-'** : points reliés par des segments
    - **'- -'** : points reliés par des segments en pointillés
    - **'-o'** : gros points reliés par des segments (on peut combiner les options)
    - **'b'**,**'r'**,**'g'**,**'y'** pour de la couleur (bleu, rouge, vert, jaune, etc...)

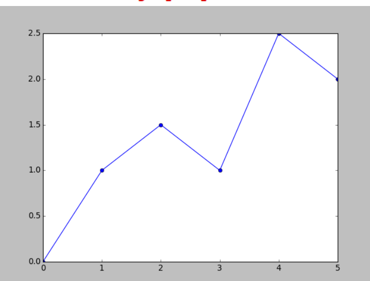
matplotlib



matplotlib.pyplot

Exemple : Pour tracer un nuage de points reliés par des segments

```
>>> absc = [0, 1, 2, 3, 4, 5] #la liste des abscisses
>>> ordo = [0, 1, 1.5, 1, 2.5, 2] #liste des ordonnées
>>> #pour créer le graphique gros points reliés par des segments
>>> plt.plot(absc, ordo, '-o')
[<matplotlib.lines.Line2D object at 0xbcc21ac>]
>>> plt.show() #pour afficher le graphique
```



matplotlib



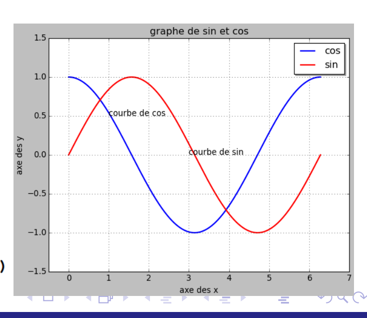
matplotlib.pyplot

Pour améliorer le graphique en ajoutant quelques fonctions :

- **axis**(L) prend une liste de L=[xmin, xmax, ymin, ymax] pour spécifier les limites des axes des abscisses et des ordonnées
- **xlabel**('Axe des x') pour afficher un label de l'axe des abscisses
- **ylabel**('Axe des y') pour afficher un label de l'axe des ordonnées
- **title**('Titre') pour afficher un titre
- **text**(x, y, 'text') pour afficher un text dans la position (x,y)
- **grid**(True) pour afficher une grille
- **legend**(('elt_légende1','elt_légende2'), 'upper right', shadow = True) pour afficher une légende en haut à droite

matplotlib

```
#Pour tracer deux courbe de sin(x) et cos(x)
import matplotlib.pyplot as plt
import numpy as np
#fonction qui retourne les valeurs sin(x),∀x ∈ axe absc
def fsin(x):
    return np.sin(x)
#fonction qui retourne les valeurs cos(x),∀x ∈ axe absc
def fcos(x):
    return np.cos(x)
#absc = axe des abscisses entre 0° et 360°
absc = np.linspace(0, 2*np.pi, 360)
#pour créer la courbe de cos(x)
plt.plot(absc,fcos(absc),'b',linewidth=2)
#pour créer la courbe de sin(x)
plt.plot(absc,fsin(absc),'r',linewidth=2)
#pour les limites spécifier des axes de x et y
plt.axis([-0.5,7,-1.5,1.5])
plt.xlabel('axe des x')#titre pour l'axe des abscisses
plt.ylabel('axe des y')#titre pour l'axe des ordonnées
plt.title('graphe de sin et cos')#titre du graphe
#un texte dans la position(x=3,y=0)
plt.text(3, 0,'courbe de sin')
plt.text(1, 0.5,'courbe de cos')
plt.grid(True)#pour afficher la grille
#pour afficher la légende
plt.legend(('cos','sin'), 'upper right', shadow = True)
plt.show()
```



matplotlib



matplotlib.pyplot

Exercices : Tracé à l'aide de matplotlib.pyplot

1. La courbe représentative de y=tan(x) pour $x \in [-\pi/2 + 0.1, \pi/2 - 0.1]$
2. Les courbes représentatives de y=ln(x) et de y=sin(x)/x pour $x \in]0, 10]$
3. La courbe paramétrée $x(t) = t.\cos(t)$; $y(t) = t.\sin(t)$ pour $t \in [0, 10]$

matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
#---------------------ex1-------------------------
Xa = np.linspace(-np.pi/2 + 0.1, np.pi/2 - 0.1, 100)
plt.figure(1)
#plt.subplot(311)
plt.plot(Xa,np.tan(Xa),'y',linewidth=2)
plt.text(0,1, 'tan')
plt.title('ex1')
#---------------------ex2-------------------------
Xb = np.linspace(1,10,1000)
#Xb = Xb[1:]
plt.figure(2)
#plt.subplot(312)
plt.plot(Xb,np.log(Xb),'b',Xb,np.sin(Xb)/Xb,'r')
plt.title('ex2')
plt.legend(('ln(x)', 'sin(x)/x'), 'bottom')
#---------------------ex3-------------------------
T = np.linspace(0,10,100)
X = T*np.cos(T)
Y = T*np.sin(T)
plt.figure(3)
#plt.subplot(313)
plt.plot(X,Y)
plt.title('ex3')
plt.grid(True)
plt.show()
```

Résolution d'équation algébrique



$f(x) = 0$

On peut toujours écrire une équation sous la forme : $f(x) = 0$
Il faut chercher les solutions de cette équation, où f est une fonction à valeurs réelles

Méthodes proposées

L'objectif est de trouver une solution approchée. Deux méthodes vont être présentées :

1. La méthode de *dichotomie* : s'applique à une fonction continue
2. La méthode de *Newton* : fournit une vitesse de convergence bien plus élevée

Résolution d'équation algébrique

 $f(x) = 0$

On peut toujours écrire une équation sous la forme : $f(x) = 0$
Il faut chercher les solutions de cette équation, où f est une fonction à valeurs réelles

Méthodes proposées

L'objectif est de trouver une solution approchée. Deux méthodes vont être présentées :

1. La méthode de *dichotomie* : s'applique à une fonction continue
2. La méthode de *Newton* : fournit une vitesse de convergence bien plus élevée

Résolution d'équation algébrique : La méthode de dichotomie

On cherche une racine x_{sol} de l'équation f(x) = 0

Principe de la recherche d'un zéro de fonction par dichotomie

On suppose que :

1. f est une fonction continue sur un intervalle [a,b]
2. f(a) et f(b) sont de signes contraires : c.à.d. (f(a)*f(b)<0)
3. Le théorème des valeurs intermédiaires nous assure que f s'annule entre a et b

$\Rightarrow$ Existence d'au moins un zéro de f dans [a,b]



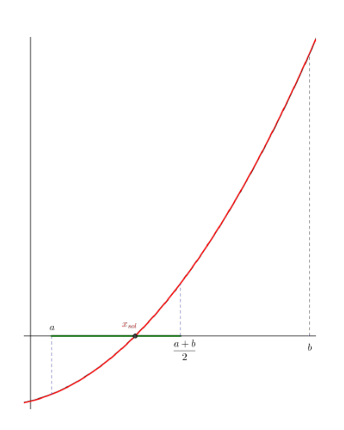
Résolution d'équation algébrique : La méthode de dichotomie

Algorithme

Soit [a,b] l'intervalle de recherche.

Démarche à répéter :

1. m = (a+b)/2
2. on calcule f(m)
3. si f(a)*f(m)<0 c.à.d. le zéro se trouve dans [a,m]
4. sinon, le zéro se trouve dans [m,b]



Résolution d'équation algébrique : La méthode de dichotomie

Terminaison de l'algorithme

Pour arrêter l'algorithme :

1. il faut fixer une marge d'erreur (ε condition d'arrêt)
2. à chaque itération ***i*** la longueur $b_i - a_i \geq 0$ est divisée par 2
3. c.à.d : on arrête l'algorithme lorsque $b_i - a_i = \frac{b-a}{2^n} \leq \varepsilon$, tel que n est le nombre d'itération
4. ou bien $(b_i - a_i) \leq \varepsilon$
5. on décide de renvoyer $\frac{b_i+a_i}{2}$



Résolution d'équation algébrique : La méthode de dichotomie

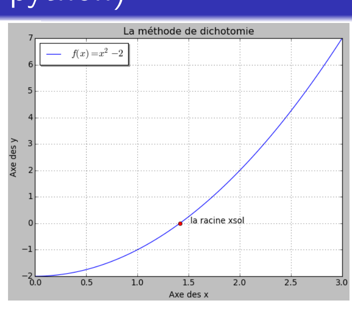
Exemple

Soit la fonction $f(x) = x^2 - 2$

1. f est continue sur l'intervalle [1, 2]
2. f(1)=-1 et f(2)=2⇒f(1)*f(2)<0 et f(x)=0 possède une solution dans [1,2]
3. $m = \frac{2+1}{2} = 1.5 \Rightarrow$ f(1.5) = 0.25, c.à.d : f(1)*f(1.5)<0 et f(x)=0 possède une solution dans [1,1.5]
4. $m = \frac{1.5+1}{2} = 1.25 \Rightarrow$ f(1.25) = -0.4675, c.à.d : f(1.25)*f(1.5)<0 et f(x)=0 possède une solution dans [1.25,1.5]
5. ...
6. f(1.41430664062)> 0> f(1.4140625)⇒f(x)=0 possède une solution dans [1.4140625,1.41430664062]
7. On arrête l'algorithme lorsqu'on atteint une valeur correspondant à la précision ε demandée

Résolution d'équation algébrique : La méthode de dichotomie (implémentation python)

```
import matplotlib.pyplot as plt
import numpy as np
def f(x):#définition de la fonction f(x)=x²-2
    return x**2 - 2
def dichotomie(f,a,b,eps):
    ai=a
    bi=b
    while bi-ai > eps:
        m=(ai+bi)/2
        if f(ai)*f(m)<0:#m sup a la racine
            bi=m
        elif f(ai)*f(m)>0:#m inf a la racine
            ai=m
        else:
            return m #m==0
    return m
X = np.linspace(0,3,100)
plt.plot(X,f(X))
plt.legend(('$f(x)=x^2-2$',),'upper left',shadow=True)
plt.title('La méthode de dichotomie')
plt.xlabel('Axe des x')
plt.ylabel('Axe des y')
xsol=dichotomie(f,1,2,0.00001)
plt.plot(xsol,f(xsol),'or')#pour afficher la racine
plt.text(xsol+0.1,f(xsol),'la racine xsol')
plt.grid(True)
plt.show()
```



Résolution d'équation algébrique : La méthode de Newton

Principe de la recherche d'un zéro de fonction par La méthode de Newton

Soit [a,b] l'intervalle de recherche, on suppose que :

1. f est dérivable sur [a, b]
2. la fonction dérivée f' ne s'annule pas sur [a, b]

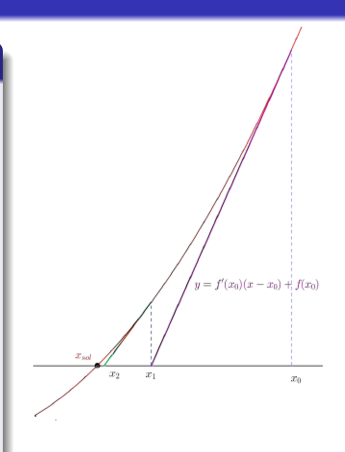
$\Rightarrow$ On veut construire une suite $(x_n)_{n\in\mathbb{N}}$ convergente vers x_{sol}

Résolution d'équation algébrique : La méthode de Newton

Principe La méthode de Newton

cette méthode considère la suite définie par son premier terme $x_0 = b$ et par la relation : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, cette suite obtenue par cette démarche :

- on considère alors la tangente à la courbe de f au point d'abscisse $x_0 = b$, qui a pour équation $y = f'(x_0)(x - x_0) + f(x_0)$
- le point d'intersection de cette droite et de l'axe des abscisses permet d'approcher x_{sol}
- on note x_1 vérifiant $f'(x_0)(x_1 - x_0) + f(x_0) = 0$, c.à.d que l'on pose $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$
- Puis on recommence : le réel x_n étant construit, on pose : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$





Résolution d'équation algébrique : La méthode de Newton

Principe La méthode de Newton

- pour arrêter les calculs, on prend comme critère d'arrêt (ε condition d'arrêt), tel que : $x_{n+1} - x_n \leq \varepsilon$
- Il faut savoir calculer la fonction dérivée, et avoir démontré que pour une valeur x_0 la suite (x_n) est bien définie (pas de division par zéro)

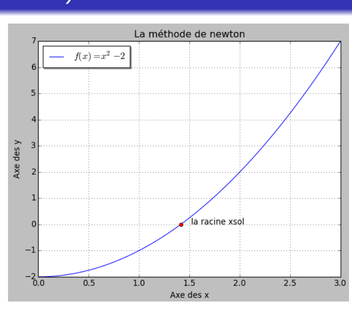
Résolution d'équation algébrique : La méthode de Newton (implémentation python)

```
import matplotlib.pyplot as plt
import numpy as np
def f(x):#définition de la fonction f(x)=x²-2
    return x**2 - 2

def fp(x):#dérivée de f : f'(x)=2x
    return 2*x

def newton(f,fp,x0,eps):
    x1 = x0 - f(x0)/fp(x0)
    while abs(x1-x0) > eps:
        x0 = x1
        x1 = x0 - f(x0)/fp(x0)
    return x1

X = np.linspace(0,3,100)
plt.plot(X,f(X))
plt.legend(('$f(x)=x^2-2$',),'upper left',shadow=True)
plt.title('La méthode de dichotomie')
plt.xlabel('Axe des x')
plt.ylabel('Axe des y')
xsol=newton(f,fp,2,0.00001)
plt.plot(xsol,f(xsol),'or')#pour afficher la racine
plt.text(xsol+0.1,f(xsol),'la racine xsol')
plt.grid(True)
plt.show()
```



Résolution d'équation algébrique : Les fonctions prédéfinies de la bibliothèque scipy

Fonction prédéfinies de dichotomie et de newton

- python dispose de bibliothèques de fonctions scientifiques prédéfinies
- la bibliothèque **scipy.optimize** contient la méthode de dichotomie et de newton :
  1. la fonction **bisect** qui implémente la méthode de dichotomie
  2. la fonction **newton** qui implémente la méthode de dichotomie

```
import scipy.optimize as sp
:            :            :
sp.bisect(f, borne_inf, borne_sup)
:            :            :
sp.newton(f, position_depart, derivée_f)
```

Résolution d'équation algébrique : Les fonctions prédéfinies de la bibliothèque scipy



scipy.optimize

Exercice :chercher la racine x_{sol} de l'équation f(x) = cos(x) en utilisant :

1. la fonction **bisect** $\forall x \in [0, \pi]$
2. la fonction **newton** $\forall x \in [\pi/100, \pi/3]$

```
>>> import scipy.optimize as sp
>>> import numpy as np
>>> def f(x):#f(x)=cos(x)
        return np.cos(x)

>>> def fp(x):#f'(x)=-sin(x)
        return -np.sin(x)

>>> sp.bisect(f,0,np.pi)#dichotomie
1.5707963267956109
>>> sp.newton(f,np.pi/3,fp)#newton
1.5707963267948966
```

Résolution d'équation algébrique : (Racines d'un polynôme)

Fonction prédéfinie **roots**

La fonction **roots** de la bibliothèque **numpy** détermine les racines dans $\mathbb{C}$ d'un polynôme

```
>>> import numpy as np
>>> P1=[1,0,-2] #x**2-2
>>> racines = np.roots(P1)
>>> racines
array([-1.41421356,  1.41421356])
>>> P2=[1,1,1]#x**2+x+1
>>> racines = np.roots(P2)
>>> racines
array([-0.5+0.8660254j, -0.5-0.8660254j])
>>>
```